Stealthy Information Leakage from Android Smartphone through Screenshot and OCR

Yeon-kyung Kim
Department of Computer Engineering
Hannam University
Deajeon, Republic of Korea
kimyeonkyung.kr@gmail.com

Man-Hee Lee
Department of Computer Engineering
Hannam University
Deajeon, Republic of Korea
manheelee@hnu.kr

Han- Jea Yoon
Department of Computer Engineering
Hannam University
Deajoen, Republic of Korea
hanjae.karoha@gmail.com

Abstract—A large number of malicious apps focus on stealing personal and financial information. It is very important to detect such apps as early as possible in order to prevent subsequent crimes. DroidBox incorporating TaintDroid detects leakage of private information, IMEI (International Mobile Station Equipment Identity) and IMSI (International Mobile Subscriber Identity). DroidBox covers many possible information leakage paths, but it is also known that there are several paths to leak the information without detection. We utilized one attack path, screen bitmap memory, in order to propose a collection system that retrieves IMEI and IMSI information through screenshot image and extracts the information from the image by OCR (Optical Character Recognition) automatically. Furthermore, we found out that *sans* font showed very low recognition rate while *serif* and *mono* showed relatively high recognition rate. We also hid a screenshot activity from users. Therefore, the proposed method can be used to leak any information without worry of detection by DroidBox, users, text-based packet inspections tools.

Keywords-DroidBox; TaintDroid; Taint; Android Application analysis; Anti-taint;

I. INTRODUCTION

With increase of smart phone usage, malicious apps also increase very rapidly. Smart phone can be characterized by continued network connectivity, software openness, and open app store, thus making smart phone usage environment vulnerable to various security attacks. Mobile apps fall into several categories: send SMS, malfunction phones, use up battery, leak private and financial information, and so on. Among various threats, information leakage threat grows more critical because people store almost everything from personal information to business data as BYOD (Bring Your Own Device) concept becomes more common.

Traditionally there are some methods to detect or prevent such information leak malware. First, static analysis investigates into execution codes without running the code to see if application under investigation includes executable paths to leak information. Second, in dynamic analysis, application is run under test environments such as a virtualized system where every single action of the application can be recorded or monitored for further analysis. Additionally, DRM (Digital Rights Management) or DLP (Data Leakage Prevention) solutions can be applied to track or detect how interesting data is used or moved.

Due to the rapid increase of malware, dynamic analysis that can be done automatically receives more attention these days. Among many techniques used for dynamic analysis, tainting analysis gives security analysts new opportunities by providing detailed reports how data in the system is used and processed. TaintDroid is almost the first successful environment to analyze Android app with flow analysis functionalities [3], and DroidBox incorporating TaintDroid provides more useful functionalities [4].

G. Sarwar *et al.* presented many side channel attack methods to avoid flow analysis based detection [1]. In our previous study, we performed a feasibility test for utilizing one of the techniques, called bitmap text, to construct a large scale private information collection system [6]. In bitmap text skill, a malicious app first prints private information on the screen as text format, then captures a screenshot by accessing screen bitmap memory. Then, it sends out the image to a predestined host, Since TaintDroid does not provide tainting analysis on the bitmap memory, this method can go unnoticed.

In this research, we implemented the proposed system. In addition, we found out that character recognition rates vary according to font types. As an engineering issue, we tested three fonts, *sans*, *serif*, and *mono*, provided as default fonts by Android system. *sans* font showed very a low recognition rate while *serif* and *mono* showed a relatively high recognition rate only with font size bigger than 12.

Finally, in some smart phones, we successfully hid a screenshot activity from users. To capture a screenshot, the information should be stored at screen bitmap to be shown to users, inevitably resulting in user awareness of leaked information. We used a simple technique to hide this action from users by using a short delay between the activity of printing texts on screen and subsequent activity. In our experiments, 0.5 second is good enough both for successfully capturing a screen image and for not showing any texts on the screen. In conclusion, our method can help retrieve any text information from smart phones without being detected by DroidBox, users, and text-based packet inspection tools.

II. Related Work

In this section we give a brief overview of DroidBox, side channel attack for DroidBox, and private information collection system that we proposed in our previous study [6].

A. *DroidBox*

DroidBox is a dynamic analysis tool for Android application based on Linux, and Android kernel 4.1 version is modified for DroidBox [4]. Also it analyzes the malicious applications by using sandbox and tainting techniques based on TaintDroid [3]. When DroidBox completes the analysis of malicious applications, it provides analysis results with a Json file. This includes network communications, file read/write operations, information leakage, encryption module operations, etc. It keeps tracking private information, IMEI and IMSI, to see if the information is processed and sent outside of the devices, called data sinks including file, network connection, and SMS.

B. *Diverting DroidBox Information Leakage Detection Scheme*

G. Sarwar *et al.* presented several methods to bypass flow analysis based detection schemes of DroidBox [1]. First, control dependence is to copy tainted data without direct assignment. Examples are simple encoding attack, Count-to-X attack, and deliberate exception attack. Second, trusted benign codes are used to divert flow analysis. The main idea of this category is to utilize normal system commands and files that are not traceable inside the code. The authors introduced a system command attack and a system-file hybrid attack. Finally, this paper explained several side channel bypass techniques including the bypass timing attack, file length attack, clipboard length attack, bitmap cache attack, text scaling attack, direct buffer attack, and remote control channel attack.

C. *Private Information Collection System (PICS)*

By using the fact that information leakage via screen bitmap image is not detected by antivirus solution or DroidBox, we proposed a private information collection system in a simple client-server model as shown in Figure 1. Client apps that we developed for this system collect IMEI value from mobile devices through GetDeviceID() method of *TelephonyManager*, gains its bitmap image using Drawing cache and getDrawingCache, and sends the images to a main server. The main server receives and stores the images and run an OCR module to recover IMEI values from the images.

Figuer 2 is a part of DroidBox result generated when our app sent an IMEI value to the main server. *opennet* and *sendnet* sections show that some data was sent to a host with 203.247.39.97 IP address and port number 22, but *dataleaks* section shows nothing, meaning that there is no information leakage. Therefore, we can confirm that it is possible to bypass information leakage detection of DroidBox as explained by [1].

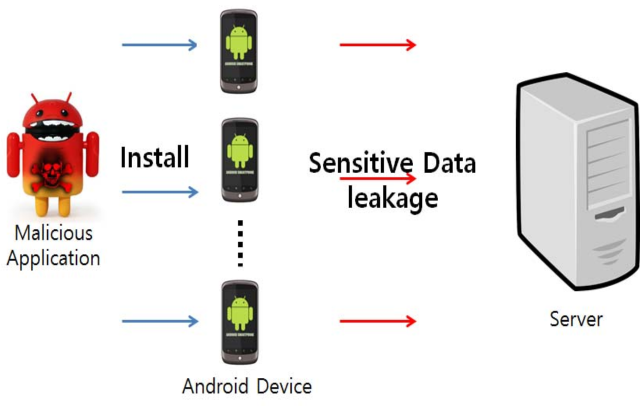

FIGURE 1. SYSTEM DESIGN

```
"dataleaks": {},
........
"opennet": { "2.901952028274536": {"desthost": "203.247.39.97", "destport": "22", "fd": "16" } },
"recvnet": { "3.098494052886963": { "data": "53", "host": "Server IP", "port": "22", "type": "net read" },}},
........
"sendnet": { "3.011586904525757": {"data": "5353482d322e302d4a5343482d302e312e35320a", "desthost": "Server IP", "destport": "22", "fd": "16", "operation": "send", "type": "net write"}}
........
```

FIGURE 2. DROIDBOX RESULT

III. PICS IMPLEMENTATION AND FONT SELECTION

In this research, we first implemented PICS′s server. Figure 3 shows detailed architecture of PICS. Server consists of several modules including sftp server, DBMS, and an OCR module. For field deployment of PICS, we can modify the current architecture such as inserting an intermediate server layer. If a server IP address or a domain name is inserted in the binary code, the existence of the server can be revealed and taken down easily. To increase anonymity of client-server communication, the intermediate layer can utilize well-known server′s blogs where client apps can post image files so that the server visits those blogs later for collecting the images.

We use *pytesseract* OCR module written in python[5]. At first, we expected character recognition rates do not differ from font types, but we figured out there are big recognition variations depending on font types and sizes. So, we tried to find which combination of font and size is the best for PICS. We use three fonts (*sans*, *serif*, *mono*) provided by Android system as default fonts and increase font size from one to 20. We performed three tests per a combination of one font type and one font size, totally performing 1,800 tests.

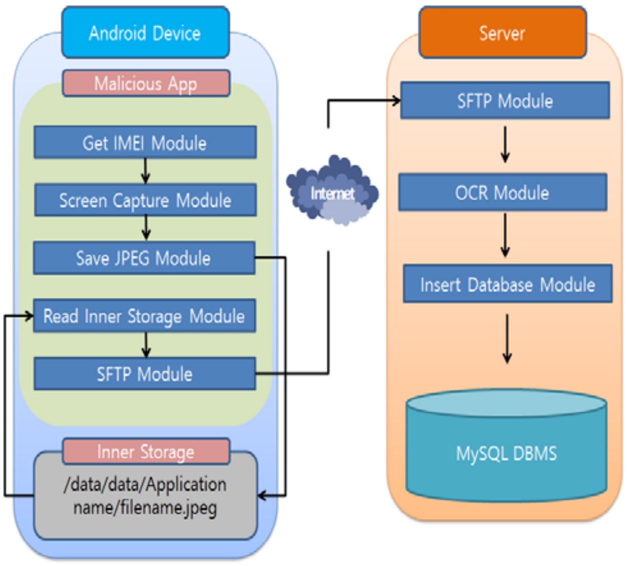

FIGURE 3. DETAILED ARCHITECTURE OF PICS

Figure 4 is a result of OCR rates of *sans* font. *sans* font becomes recognizable from font size 13 (3.33%), and shows the highest recognition rate at font size 19 (23.33%).

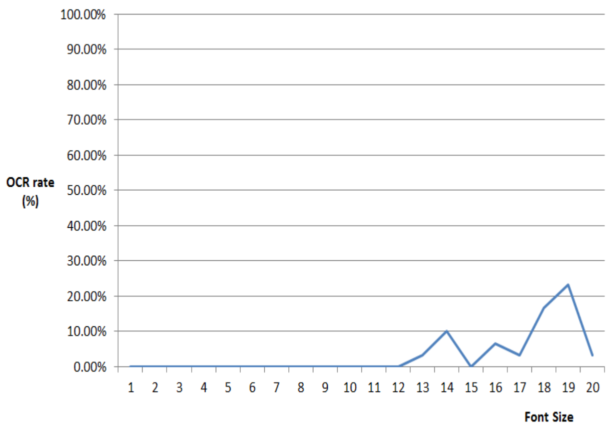

FIGURE 4. *SANS* OCR RATE

As shown in Figure 5, the OCR module begins to recognize *serif* font from font size 10 (26.67%), and shows the best recognition rate between 14 and 17, and at 20.

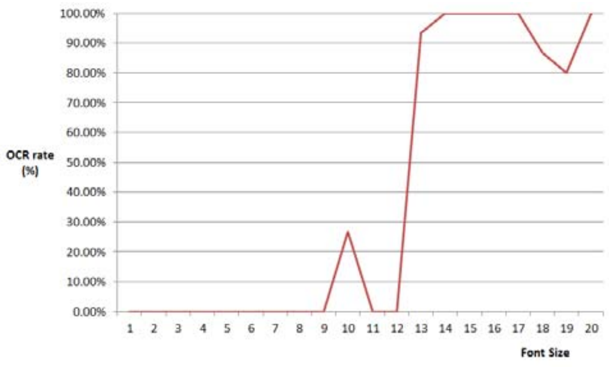

FIGURE 5. *SERIF* OCR RATE

mono font begins to be recognized from font size 9 (6.67%), and shows very high recognition rates from 14 to 18.

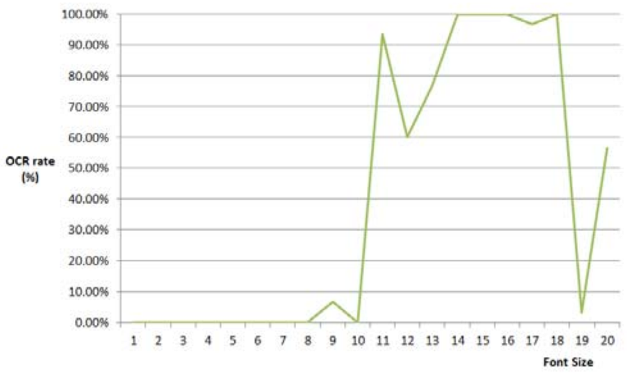

FIGURE 6. *MONO* OCR RATE GRAPH

In summary, *sans* font showed very low recognition rates while *serif* and *mono* showed relatively high recognition rates only with font size bigger than 12. Another interesting finding is that the recognition rate does not increase linearly as the font size increases. Since OCR recognition algorithm is out of our research scope, we did not seek for its reason, but we suppose that due to each algorithm's characteristics the bigger font size do not always ensure the better recognition rate. Therefore, before constructing PICS for real life, detail recognition tests should be preceded.

IV. VISUAL DETECTION AVOIDANCE

Till now, we showed that it is hard to detect information leakage using screenshot, but an application user can easily do so because some texts possibly including bank and personal information will be shown on the screen and disappear shortly.

To solve this problem, we found a trick to take a screenshot without showing anything on the screen. We use an Android Activity that generates a full screen window. We found that displaying a window needs some delay before the window appears to users on the screen, and when a new next activity is executed shortly after the previous activity, the previous activity failed to appear on the screen. Luckily enough, accessing bitmap memory and retrieving screen image are possible.

In Figure 6, imei.setText() method makes this window shown on the screen with IMEI information written in the window. screenshot(capView) method captures the bitmap image screenshot. For preventing the previous window from appearing, we use a dummy intent *i* to make a new activity on the screen. By executing startActivity(*i*) after 0.5 second, we successfully covers up the first window.

Please note that this trick does not always work for all types of smart phones. Probably it depends on hardware or software components. For this research, we tested Galaxy S and Galaxy Nexus, and 0.5 second delay works very well.

```
........
//setting IMEI in textbox
imei = (TextView) findViewById(R.id.textView2);
imei.setTextColor(Color.BLACK);
imei.setTypeface(str_font);
imei.setTextSize(cnt_size);
imei.setText(global_imei_string);
........
capView = getWindow().getDecorView();
screenshot(capView);
........
Intent i = new Intent(MainActivity.this, FakeActivity.class);
try {
    Thread.sleep(500);
    startActivity(i);
    finish();
    }
    catch (Exception e) {
}
........
```

FIGURE 7. IMEI VALUE EXPOSURE PREVENTION TO THE USER

V. CONCLUSION

In this research, we proposed and implemented a private information collection system stealthily retrieving data via screens bitmap images. DroidBox, a de facto standard dynamic flow analysis tool, cannot detect this leakage since the screen bitmap is out of DroidBox's tracing scope. In constructing the server, we extract text information from the image by OCR. We also found out that *sans* font showed a very low recognition rate while *serif* and *mono* showed a relatively high recognition rate only with font size bigger than 12. In some smart phones, we successfully hid a screenshot activity from users. Therefore, the proposed method can be used to leak any information without worry of detection by DroidBox, users, and text-based packet inspection tools.

We are currently investigating into how DroidBox traces private information even to bitmap memory and a static analysis method to detect such Android apps.

REFERENCES

[1] G. Sarwar(Babil), Olivier Mehani, Roksana Boreli and Mohamed Ali kaafar, "On the Effectiveness fo Dynamic Taint Analysis for Protecting Against Private Information Leaks on Android-based Devices," NICTA Technical Report RT-7091, 2013

[2] S. Mori, CY Suen, K. Yamamoto, "Historical review of OCR research and development," IEEE 80(issue, 7), 1922, 1029-1058

[3] Willian Enck, Peter Gilbert, Byung-Gon Chun, Landon P.Cox, Jea-Yeon Jung, Patrick McDaniel and Anmol N.Sheth, "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones," OSDI vol. 10, October 2010, pp.225-270

[4] Michael Spreitzenbarth, Thomas Schreck, Florian Echtler, Daniel Arp, Johannes Hoffmann, "Mobile-Sandbox: combining static and dynamic analysis with machine-learning techniques," DOI 10.1007/s10207-014-0250-0, 2014

[5] Pytesseract, https://pypi.python.org/pypi/pytesseract/0.1

[6] Yeon-Kyung Kim, Han-Jea Yoon, Man-Hee Lee, "Experiment of Side Channel Attack for Diverting Data leak Detection by Taint Analysis of DroidBox," CISC-S'15, Jun 2015

[7] Lantz. P, "The Honeynet Project," http://www.honeynet.org

[8] Lantz. P, "An Android Application Sandbox for Dynamic Analysis," Maters Thesis, Department of Electrical and Information Technology, Sweden, Lund University, 2011

[9] Michael Spreizenbarth, Florian Echtler, Johannes Hoffmann, "Mobile-Sandbox: Having a Deeper Look into Android Applications," SAC'13, March 2013