Android Applications: Data Leaks via Advertising Libraries

Veelasha Moonsamy
School of IT
Deakin University
Melbourne, Australia
Email: v.moonsamy@research.deakin.edu.au

Lynn Batten
School of IT
Deakin University
Melbourne, Australia
Email: lynn.batten@deakin.edu.au

***Abstract*—Recent studies have determined that many Android applications in both official and non-official online markets expose details of the users' smartphones without user consent. In this paper, we explain why such applications leak, how they leak and where the data is leaked to. In order to achieve this, we combine static and dynamic analysis to examine Java classes and application behaviour for a set of popular, clean applications from the Finance and Games categories. We observed that all the applications in our data set which leaked information (10%) had third-party advertising libraries embedded in their respective Java packages.**

I. Introduction

Android, the Google-owned leading smartphone platform [1], allows users to download applications from both the official and non-official markets. However, only the applications from the official market, Google Play, are vetted before they are available for download. On the other hand, the applications available on the non-official markets are managed by individuals and businesses and are not tested for presence of malware. Nevertheless, Android users are largely attracted to non-official markets as they offer a variety of applications which are not available elsewhere. The absence of security checking has not deterred users from downloading applications from non-official markets. In fact, the numbers of applications available and the numbers downloaded from both types of markets are soaring.

In order to maintain such an uptake in applications, developers are encouraged to market their products free of charge. However, this action can only lead to a feasible business model as long as the developers embed advertising in their applications. In fact, the more advertising libraries they embed in their applications, the higher the revenue. In-application advertisements connect to the core level of the platform and are difficult to remove once the application is installed. According to [2] and [3], advertisements can send out sensitive information such as the International Mobile Equipment Identity (IMEI) code identifying the mobile device, the International Mobile Subscriber Identity (IMSI) number found in the sim card and users' physical location without their knowledge.

While malicious applications are expected to leak identifying device information to external parties without user permission, it would be concerning that applications classified as non-malicious do so. Therefore, the aim of the paper is to study a set of clean applications and analyse them to see if they leak any sensitive data. We formally define the following three terms, malicious, clean and leaky applications, that are used frequently throughout the paper. A **Malicious** application is considered as an application designed to harm a computer or software it is running. Such behaviour can be detected by anit-virus software products. A **Clean** application is an application which has not been identified as malicious. A **leaky** application has the capability of providing personal information to a third-party without the knowledge or permission of the user.

The paper is organised as follows: in Section II, an overview of the Android platform is provided; this is followed by, a summary of related work in Section III. In Section IV, we give details on the dataset collection and experimental work. Finally, we discuss our empirical results and draw conclusions in Section V.

II. Background

In this section, we provide a brief background on the Android platform and its related components.

A. Android Platform

Android is a Linux-based operating system (OS) and is made up of multiple layers consisting of the OS, the Java libraries and a set of basic built-in applications. Additional applications can be downloaded for free or for a small price.

Google provides developers with a Software Development Kit (SDK), including a collection of Java libraries and classes, sample applications and relevant documentations, to assist them with new application development. The SDK can be used as a plug-in for Eclipse IDE and allows developers to code their applications in a rich Java environment.

An Android application is made up of the Class and Resources folder and the AndroidManifest.xml file. The Class folder contains the application's Java source code; the Resources folder stores the multimedia files; the AndroidManifest.xml file lists the permissions which are declared by the developer.

B. Android Permissions

Google applies the permission system as a measure to restrict access to privileged system resources. Application developers have to explicitly mention the permission they would like users to grant in the AndroidManifest.xml file. This file is bundled together with classes.dex, which is the optimized

version of the application Java code, and then converted into an application package file (.apk). After application download, the user is presented with an interface listing describing all the permissions that the application requires. The application is installed successfully only when the user chooses to grant access to all these permissions.

Android Permissions can be classified into four types: *Normal*, *Dangerous*, *Signature* and *SignatureorSystem*. The *Normal* permissions can be viewed after application installation and do not need user's approval. *Dangerous* permissions have access to restricted resources on the hardware and require user's approval before application installation. Permissions in the *Signature* category are granted without the user's knowledge only if the application is signed with the device manufacturer's certificate. *SignatureorSystem* permissions are assigned only to the applications that are in the Android system image or are signed with the device manufacturer's certificate. More information on the Android platform and permissions can be found in [4].

III. Literature Review

In this section, we provide an overview of the related work on advertising libraries and data leaks in Android applications.

In-application advertising is one of the main sources of revenue for application developers. On the Android platform, advertising libraries are assigned the same permissions granted to the host application. Additionally, developers can include additional permissions which can be used to enhance targeted-advertising and consequently generate more revenue. As a result, this in turn introduces vulnerabilities within the application, causing data leaks via advertising. We present some of the existing work that investigates the possibility of separating the privileges of advertising libraries on the Android platform.

Pearce et al. [3] considered separating the privilege of the host application from that of the embedded advertising library. In order to so do, they introduced a new set of API calls and permissions to be used only within advertising libraries. The authors pointed out that this change in the Android framework would not require application developers to embed advertising libraries anymore; instead, the new set of API calls and permissions would be used to instruct the applications to fetch the advertisements from certain sites. Additionally, the proposed technique would eliminate the need for an application to request unnecessary permissions, thus ensuring that advertising libraries execute properly.

Shekhar et al. [5] looked into the idea of splitting the host application and advertising into separate processes. In other words, the advertising libraries and host application would not be assigned the same permissions. In their framework, the authors also examined the extent to which applications use their permissions only for advertising purposes. The results showed that permissions such as INTERNET and READ_PHONE_STATE are widely used only for advertisements.

Grace et al. [6] collected a dataset of 100,000 applications from the official market and manually extracted the AndroidManifest.xml files in order to separate the applications that used the INTERNET permission. They manually investigated a set of approximately 52,000 applications with INTERNET permission to determine if the advertising libraries requested dangerous permissions and to establish their impact on the related API calls. In contrast, in our work, we first execute the applications in our dataset and then, by examining the log files and AndroidManifest.xml files, we are able to deduce if an application leaked device-related information via advertising libraries.

In [7], Gibler et al. proposed a framework to detect leaks of personal information. They started by generating a permission map, which included information about API calls and the related permissions which they require to execute; the map also contained information about potential sources of leaks. Then, they used the decompiled version of an application to generate a call graph. They iterated repeatedly to cover all possible execution paths of the application and recorded the instances where external methods invoke restricted information; hence, they identified potential leaks and their types. After evaluating their work on a set of 23,000 applications which were collected from two different non-official markets, they found 9631 possible privacy leaks in 3,258 applications.

As for the work of Chan et al. [8], the authors parsed the AndroidManifest.xml files to collect the requested permissions and then identified components, such as activity and broadcast receiver, that are potential sources of data leaks. They then applied inter-procedural control flow searching for each component and followed the information flow to confirm if the leaks actually occurred. The work by authors of both [9] and [10] investigated the occurrence of data leaks within the Dalvik bytecode implementation. They began by generating a reduced set of execution instructions in order to capture the relevant information flow paths. Based on the aforementioned set, the authors then manually traversed each application, written in bytecode, to identify potential sources of leaks.

While the authors of [8]–[10] identify leaks, and what is leaked, in this paper, we go further and determine how the leaks occur and the destinations of the leaked data. In our work, we reverse-engineer the application package files and statically analyse the Java version of the applications in order to determine the occurrence of a leak and its possible cause(s). Moreover, we further support our experiment by dynamically executing the applications from our dataset in a sandbox to monitor and record the behaviours between an application and the operating system during run-time, this allows us to identify the leaked information as well as its destination.

IV. Experimental Work

In this section, we describe our dataset collection and further explain our experimental set-up and dataset execution.

A. Dataset Collection

For our dataset, we collect popular clean applications under the Finance and Games category which are classified as non-malicious. It is expected that malicious applications leak, depending on the extent of the harmful actions they have been instructed to carry out; hence, they are of no interest to this experiment. Additionally, we only collect those applications which include the INTERNET permission as it is highly used in free applications and is also one of the main facilitators of

information leaks. We upload each application to VirusTotal to confirm that it is clean. The numbers and source are shown in Table I.

TABLE I. TOTAL NUMBER OF APPLICATIONS COLLECTED

	Number of Applications
Google Play	49 Finance and 24 Games
SlideME	50 Finance
Total	123 applications

B. Experimental Framework

The experiment begins with a dynamic analysis test to identify any leaky applications in our dataset, followed by a static analysis. We use a research laptop equipped with Intel (i7) CPU 2.7 GHZ, 8 GB of DDR3 RAM and 720 GB hard disk on windows 7. We then install the virtual machine, VMware Workstation build-591240, which runs an Ubuntu 11.10 32bit operating system. Next, we set up the Android Emulator, along with DroidBox [11] and disable all interaction between the virtual and local host in order to build a safe environment in which to run the applications and record the execution process.

In order to determine if an application leaked during the dynamic analysis, we parse the log files and search for the section where a leak is recorded. DroidBox keeps track of five types of information when documenting a leak in the log file. These include the source of the leak (also referred to as the sink), the destination of the leak, the port number through which the information is sent to an external party, the name of the taint tag and the html-encoded data which is leaked through the GET command. As an example, a leak in one file shows up as

```
Sink: Network;
Destination:
intuitandroidtaxstatprod.122.2o7.net;
Port: 80;
Tag: TAINT_IMSI;
```

indicating that the application leaked the IMSI to an external server with address *intuitandroidtaxstatprod.122.2o7.net* via the network sink.

Of the 123 applications in our dataset, we found a total of 13 which leaked device-related information, IMEI and IMSI, to external networks. By examining the individual AndroidManifest.xml files for each leaky application to determine the list of permissions that were requested, we found that all 13 leaky applications include both the INTERNET and READ_PHONE_STATE permissions. An additional 23 applications contained both of these permissions, while we did not see leaks; this is likely because our cut-off time of 3 minutes for execution pre-empted such a leak.

Next, we use the destination addresses, recorded under the information leakage section within the log files, to search through the Java classes and compare the namespace to confirm if the application developer has made use of any external in-application libraries. This led to us to observe that all the 13 applications had third-party advertising libraries embedded in their respective Java packages. We analyse those 13 applications further in Section V.

V. DISCUSSION AND CONCLUSION

After dynamically executing a dataset of 123 clean applications, we found that 13 applications leaked information through the embedded advertising libraries. In this section, we analyse and discuss this set of 13 applications and provide some recommendations to conclude the paper.

A. Analysis of Results

Upon further analysis, we noticed that 9 out of those 13 applications included on average two additional third-party advertising libraries, excluding the one through which they leaked. Moreover, we also found a total of 9 advertising libraries embedded in one leaky application which is classified under the Games category and as many as 3 advertising libraries included, at one time, in a leaky application from the Finance category.

Generally, it is well-known that application developers earn revenues from in-application advertisements, hence, providing them with the incentive to market their applications free of charge. In fact, the more advertising libraries they embed in their applications, the higher the revenue. It is also worth mentioning that each advertiser usually has their own set of advertising libraries which can be obtained after signing up with the advertising company. Application developers do not have to fully comprehend the advertising code as they only need to follow the instructions given by the advertising companies to successfully include advertisements in their applications; therefore they may unknowingly leak sensitive information through the advertising libraries.

Advertising libraries, by default, require the application developer to include the INTERNET permission in the AndroidManifest.xml file so that the advertising company can track the number of clicks which will then be used to determine the revenue to be paid to the application developer. This leads us to conclude that the application developers do not follow the 'least-privilege' method when requesting permissions for their applications.

Furthermore, we noted that 10 leaky applications made use of the WebView class and APIs to embed the in-application advertisements. WebView can be regarded as an in-application browser and enhances the advertising display to allow the application to present web content in the advertisements. The flaw of this type of advertisement implementation is that it offers a gateway to external parties through the WebView browser to initiate an attack and eventually take control of the device, as explained in [12]. Additionally, in order for the WebView components to function correctly, the application must request the INTERNET permission during the installation thus, opening a pathway for possible web attacks that can be routed through WebView browser to the application.

B. Conclusion and Recommendations

Some device users are indeed happy to have advertisers track their location in order to be provided with a customised advertising profile and targeted advertising. On the other hand, many users feel that leakage of device ID data is an invasion of privacy (and in some cases may even be illegal) and do not want their location tracked. We believe that a user should

have the option. We recommend that every application be equipped with the functionality of permitting the device user to acquire a downloaded application for a fixed trial period (as determined from the date of the download) without tracking, and at the completion of this period, be asked to opt in to location tracking, with the option of not doing so.

We also recommend that anti-virus software developers include in their products identification of applications containing advertising libraries, or, since this is likely to be by far the majority of applications, those which do not; this flags the presence of advertising libraries to the user who may then make an educated decision to opt in or out of location tracking.

In making these recommendations, we note that the authors of [3] have also made recommendations for changes in the Android framework to address the issue of private data leaks. We believe that our solution would be easier and cheaper to implement than theirs.

REFERENCES

[1] IDC. Smartphone OS Market Share, Q1 2014. IDC. http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

[2] W. Enck, P. Gilbert, B. Chun, L. Cox, J. Jung, P. McDaniel, and A. Sheth, "TaintDroid : An information-flow tracking system for real-time privacy monitoring on smartphones," in *In Proceedings of the 9th USENIX conference on Operating systems design and implementation (OSDI 2010)*, Vancouver, Canada, October 2010, pp. 1–15.

[3] P. Pearce, A. Felt, G. Nunez, and D. Wagner, "AdDroid: Privilege Separation for Applications and Advertisers in Android," in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security (ASIA CCS 2012)*, Seoul, Korea, May 2012, pp. 1–11.

[4] V. Moonsamy, J. Rong, and S. Liu, "Mining permission patterns for contrasting clean and malicious android applications," *Future Generation of Computer Science*, Accepted September 2013 2013, 10.1016/j.future.2013.09.014.

[5] S. Shekhar, M. Dietz, and D. Wallach, "AdSplit: Separating smartphone advertising from applications," in *In Proceedings of the 20th USENIX Security Symposium (USENIX Security 2012)*, Bellevue, USA, August 2012, pp. 1–15.

[6] M. C. Grace, W. Zhou, X. Jiang, and A. Sadeghi, "Unsafe exposure analysis of mobile in-app advertisements," in *In Proceedings of the 5th ACM conference on Security and Privacy in Wireless and Mobile Networks (WISEC 2012)*, Arizona, USA, April 2012, pp. 101–112. [Online]. Available: http://doi.acm.org/10.1145/2185448.2185464

[7] C. Gibler, J. Crussell, J. Erickson, and H. Chen, "AndroidLeaks: Automatically Detecting Potential Privacy Leaks in Android Applications on a Large Scale," *Trust and Trustworthy Computing*, vol. 7344, pp. 291–307, 2012.

[8] P. Chan, L. Hui, and S. M. Yiu, "DroidChecker: Analyzing Android Applications for Capability Leak," in *In Proceedings of the 5th ACM conference on Security and Privacy in Wireless and Mobile Networks (WISEC 2012)*, Arizona, USA, April 2012, pp. 125–136.

[9] C. Mann and A. Starostin, "A framework for static detection of privacy leaks in android applications," in *In Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC 2012)*, Trento, Italy, March 2012, pp. 1457–1462.

[10] J. Kim, Y. Yoon, K. Yi, and J. Shin, "ScanDal: Static Analyzer for Detecting Privacy Leaks in Android Applications," in *In Proceedings of the 2012 Mobile Security Technologies (MoST 2012)*, California, USA, May 2012, pp. 1–10.

[11] P. Lantz, "An android application sandbox for dynamic analysis," Master's thesis, Lund University, 2011.

[12] T. Luo, H. Hao, W. Du, Y. Wang, and H. Yin, "Attacks on webview in the android system," in *In Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC 2011)*, Florida, USA, December 2011, pp. 343–352.