The Risk of Advertizing Libraries on Android Mobile Devices

Lynn Margaret Batten* and Veelasha Moonsamy*

**School of Information Technology, Deakin University, Melbourne, Australia*

lmbatten@deakin.edu.au

v.moonsamy@research.deakin.edu.au

I. INTRODUCTION

Applications for mobile devices need to be managed by the device user. The level of user management depends on the platform; for instance Apple mobile devices tend to be largely managed by the company with some input from the user, while open source mobile devices, such as those using Android, leave more decision making to the user.

In all cases, *Permissions* are the standard method of providing the user with management control. The permission system acts as a measure to restrict access to privileged system resources. When a user wishes to download an application, she is presented with an interface listing and describing all the permissions that the application requires. The application is installed successfully only when the user chooses to grant access to these permissions. However, as described in [1], generally users pay very little attention to the permission description and simply click on the install button without having a proper understanding of the consequences of such an action; on the other hand, users who do read the permission descriptions are unlikely to understand the security implications of granting those permissions.

While permissions are a means to protect resources, including the user's identity, they do not safeguard information from capture by in-application advertising libraries. Application developers make use of third-party advertising libraries in order to earn revenue based on the number of downloads of their applications. The libraries, in turn, import the user's device ID and subscriber ID for user-profiling. This information helps advertisers in carrying out targeted advertising. Consequently, the higher the number of downloads, the more revenue for the application developer and advertising company. In some cases, advertising companies will then sell this identifiable information to other advertisers for marketing purposes.

In this talk, we describe our analysis of advertising libraries on mobile devices. We indicate how they can easily lead to identity theft, especially by libraries which capture the Device/Subscriber ID in order to be paid for 'hits'. We then indicate how our work can prevent such 'theft' by means of changing the identity management approach.

II. EXPERIMENTAL WORK

In our experiments, we used the Droidbox tool which is an open source dynamic analysis tool and is the extended version of TaintDroid, which was developed by Enck et al. [2]. A thorough description of Droidbox is available in the thesis of Lantz [3].

For our dataset, we collect 123 popular clean applications under the Finance and Games category which are classified as non-malicious; 73 of these were from the Google Play market while 50 were from the popular third-party market SlideME. Additionally, we only collect those applications which include the INTERNET permission; this decision is made

based on papers referred to in [4] which demonstrate that the INTERNET permission is highly used in free applications and is also one of the main facilitators of information leaks. That same paper lists the applications chosen. We upload each application to VirusTotal to confirm that it is clean.

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 [5], along with DroidBox [6] 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 the 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 data set, 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. Moreover, we found a total of 9 advertising libraries embedded in one leaky application classified under the Games category and 3 advertising libraries included in a leaky application from the Finance category.

After the above discussion, one might expect applications with no permissions to be free of these problems. However, this has been shown not to be the case in [7] where it is pointed out that an application that does not request any permission can scan the external storage directory and return a list of applications and files that can be accessed at /sdcard/. A similar exploit can be carried out on the internal memory by scanning the /data/system/packages.list folder to retrieve the list of package names of those applications installed on the device. Furthermore, in the same folder, the file *packages.xml* stores the shared permissions of all applications on the device along with their corresponding UIDs.

Also explained in [7] is that the aforementioned vulnerabilities can be exploited by enticing the user to download a decoy application which can give the attacker access to a backdoor on the victim's device, using an active remote shell to interact with the device and therefore gaining access to the data on the device. Furthermore, the absence of the INTERNET permission does not stop an attacker from exporting the captured data onto an external server. In fact, the URI_ACTION_VIEW Intent can initiate a Web browser call and

thus allow the attacker to transfer data via the Internet. This is a common technique used by advertising libraries designed to leak device ID or subscriber ID in order to carry out targeted advertising.

III. Summary and conclusion

It is well-known that application developers earn revenues from in-application advertisements; these provide them with an incentive to market their application free of charge. The more advertising libraries they embed in their applications, the higher their revenue. It is also the case that each advertiser has their own set of advertising libraries which can be obtained after signing up with the advertising company. Application developers only need to follow the instructions given by the advertising companies to successfully include advertisements in their applications; thus, many application developers unknowingly leak sensitive information through the advertising libraries.

Furthermore, the design of the Android security framework does not permit the operating system to differentiate between the permissions required by an application and those needed by an advertising library.

We suggest that a permission system should be a means for a user to give consent to applications during the installation process (rather than during the download process) in order to allow the application to access restricted resources on the mobile device. The current permission system is not fine-grained enough to preserve the privacy of a user's personal information.

References

[1] A. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android Permissions: User Attention, Comprehension, and Behavior," UC Berkeley, Tech. Rep. UCB/EECS-2012-26, 2012.

[2] Enck, W., Gilbert, P., Chun, B. G., Cox, L. P., Jung, J., McDaniel, P., Sheth, A. N. (2010). TaintDroid : An information-flow tracking system for realtime privacy monitoring on smartphones. *Proceedings of the 9th USENIX conference on Operating systems design and implementation*.

[3] Lantz, P. (2011). An Android Application Sandbox for Dynamic Analysis. Masters Thesis. Department of Electrical and Information Technology. Sweden, Lund University.

[4] Moonsamy, V.; Alazab, M. & Batten, L. (2012), Towards an Understanding of the Impact of Advertising on Data Leaks, In the International Journal Security and Networks (IJSN), Vol 7, No.3, pp.181-193

[5] Android Developers. (2012b). "Download Android Software Development Kit." from http://developer.android.com/sdk/index.html, retrieved May 2012

[6] Lantz, P. (2011b). "The Honeynet Project." from http://www.honeynet.org/gsoc/slot5.

[7] Moonsamy, V. & Batten, L. (2012), Zero Permission Android Applications: Attacks and Defences, In Proceedings of the 3rd Workshop on Applications and Techniques in Information Security (ATIS), Melbourne, Australia, November 2012