SYS: Secure Your Smartphone using Query Analyzer

Ajay V. Rane[1]; Nikita R. Shinde[2]; Aniket A. Parab[3] Shaikh[4]
[1-3] B.E. Students
[4]Project Guide
Department of Computer Engineering,
K. C. College of Engineering & Management studies & Research,
Kopri, Thane(E)-400 603,India.

***Abstract*-Smartphone and tablet users typically store a variety of personal information on their devices, including contact information, photos, SMS, and personal data used by various apps. On Android devices, the data is stored in SQLite databases which apps access by constructing and executing queries, either directly or via Android content provider API calls. Before installing an app that uses a content provider, a user must permit for the app to read and/or write the associated data. Many users grant permissions with little understanding of the risks. Even more experienced users cannot make well-informed decisions, as they are only given very coarse information about what data the app accesses. To provide users with more detailed information about how Android apps access and modify stored data, we came up with SYS, the query Analyzer. SYS analyzes app binary code, performing a lightweight static analysis to determine possible values of string variables that are incorporated into queries. SYS reports on the content providers used and the database tables/attributes accessed and/or updated, allowing users to make more informed decisions about whether to grant permissions. This paper describes SYS's design and evaluates SYS's functionality.**

Keywords - Android; Database app; Static analysis, malware;

I. INTRODUCTION

While Android platforms and apps for them are exploding now a days, there is increasing awareness that these apps can pose threats to security and privacy [1]. Many apps access personal data that is stored in databases on the Android device, such as information about the users contacts, photos, SMS, etc. App developers must declare resources that the app uses a *manifest* file and an Android app cannot be installed unless the user explicitly permits the app to access those resources. This appears to protect users from hazardous apps. Unfortunately, users know little about what the apps actually do with these permissions, so it is difficult for a user to make an informed decision as to whether to allow installation.

In order to address this problem, we have developed SYS (Secure Your Smartphone using Query Analyzer), which applies static analysis to Android app binaries in order to reverse engineer aspects of the app's interaction with the user's databases.

SYS uses a fairly fast flow-sensitive intra-procedural analysis to estimate the possible values of string variables that are used either directly or via content provider methods to construct SQLite queries. Using dedexer [3] to disassemble the dex binary code (the format in which Android apps are downloaded), SYS constructs a graph of method calls and the points in the program that call them and control flow graph slice representing relevant aspects of the program. A (slightly modified) work-list algorithm is used to associate with each node the set of possible values of string variables at that node. Since the set of possible values for a string variable is potentially infinite, a standard iterative algorithm is not guaranteed to reach a fixed point and terminate. We deliberate that for most apps and for the variables of interest here, the algorithm will reach a fixed point after a reasonably less number of iterations.

II. BACKGROUND

Android is a complete package of software for mobile devices [4], [5]. It consists of four components in a hierarchy. At the bottom, there is a Linux kernel, which provides core system services such as security, memory management, process management, network connectivity, and driver models. The next layer is a combination of two components, the *Dalvik* virtual machine (DVM) for executing Dalvik Executable format (*.dex*) files and a set of core libraries for system, graphics, database, and web surfing. The third component is an app framework providing a set of services and systems fully supporting rich app development. Above all the components, there is a set of basic apps such as a phone app included in Android SDK.

A. dex code

Dalvik Virtual Machine is a register-based machine. A frame consists of a specific number of registers, a pointer to a .dex file, and data for execution. It employs a sand box architecture in which an app has its own process running on an instance of DVM. The advantage of the architecture is that a failure of an app does not impact other apps or the system. An Android app is written in Java and translated into Dalvik executable (dex) format. During translation to dex, separated and indexed constant pools and methods are derived from original Java classes. The pools store strings, types, fields, and methods. At runtime, the constant pools can be referenced by instructions as needed. Arguments to a method correspond to registers in the method invocation frame. Some assignment and array operation instructions perform for more than one type. For example, a 32-bit move instruction works for both integer and floats. Instructions are variable length.

B. App components

There are four different types of app components used to implement an Android app. Each component has distinct purpose and conditions.

Activities:

An activity controls a single window connecting to a UI.

Services:

A service runs in the background to support long running operations. It does not provide a UI but can be started by another component.

Broadcast receivers:

A broadcast receiver is a component that receives and responds to messages from other apps.

Content provider:

A content provider provides a way of accessing data repositories on the phone for both retrieving and manipulation. To use a content provider, it has to be associated with a designated API. A basic content provider class is bundled in the Android SDK and provides services for built-in data repository such as SMS, *audio*, *video*, *images*, *contact information* etc. App developers can also define personalized content providers; these often connect to private data repositories such as a table in *SQLite* database. This paper focuses on reverse engineering dex code in order to discover how apps access databases, either direct SQLite queries or via content providers.

C. The Manifest File

Each app must have a *manifest*, which includes essential information describing the app's components. It includes needed permissions, the minimum API level required, hardware and software features used, and referenced API libraries. The manifest file, written in XML format, is grouped along with the .dex files in the app's *.apk* file. When developers distribute apps, they post the .apk file on Google Play or another repository. Consequently SYS can only take .apk files as input, not Java source.

D. Permissions

In the Android system, apps require permission to perform sensitive operations. For example, an app needs permission in order to invoke a content provider to access data created by other apps or to access built in content providers. An app that needs additional functions not provided by default, must list the needed permissions in the manifest file. When a user installs the app, the Android system will ask for the user's consent to grant the permissions listed in the manifest. If the user denies consent, the app is not installed. Apps can create custom data repositories and make them available to other apps by declaring them in the manifest.

III. STRING ANALYSIS

The goal of SYS's analysis is to determine how the app being analyzed interacts with the user's private data stored on the Android device. To that end, SYS estimates the possible values of strings representing SQLite queries and arguments to content provider methods that access databases. We anticipate using SYS to create an on-line repository of information about Android apps, which users can consult when deciding whether to install an app. The analysis needs to be efficient enough to keep up with the large number of apps that are offered to users, and must have relatively few false positives (identifications of database interactions that cannot actually happen) and very few, if any, false negatives (missing interactions that can take place.)

SYS builds on the open source dex disassembler *Dedexer* [3], which parses the dex files of the app under analysis. SYS adds modules to build a control flow graph (CFG), to optionally perform program slicing and to analyze string variables using a form of data flow analysis. Fig. 1 summarizes SYS's architecture.

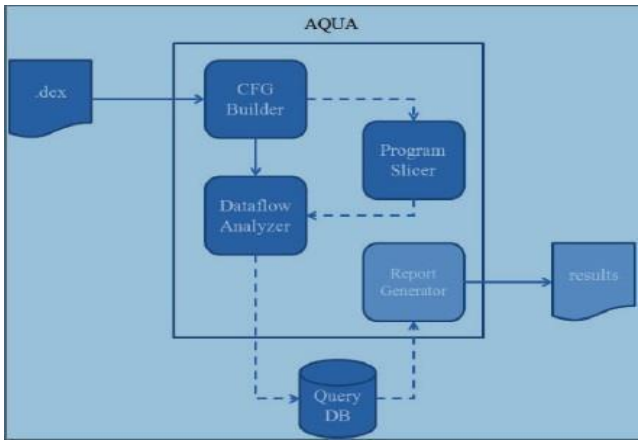

Fig. SYS Architecture

IV. LITRATURE SURVEY

There are several approaches that apply static analysis to mobile apps. Chaudhuri proposes a formal model [12] that traces flows between apps referring to permissions. In follow-on work, Fuchs et al. propose SCanDroid [13], which identifies permissions for content providers from an app's manifest, analyzes content provider uses, and checks whether data flow through identified content providers violates the extracted constraints. Both ScanDroid and SYS use data flow analysis on content providers. ScanDroid decompiles dex code into Java byte code of the apps and uses Java analysis tools, while SYS works more directly with the dex code. While ScanDroid provides information about which content providers are used, SYS provides more detailed information about how content providers are used, along with information about direct SQLite query uses. Applying the ScanDroid formal model would give SYS a capability of analyzing constraint violation. Egele et al. propose PiOS [14], which constructs controlflow graph for iOS app binaries to check whether information leaks are present by using data flow analysis. They found a number of apps on official App Store and third-party repository that leak sensitive information. Chin et al. propose ComDroid [15], which analyzes disassembled

DEX byte code to look for vulnerabilities in message passing via Intent between apps. It also employs CFG construction and data flow analysis but does not explicitly work on content provider use. Felt et al. propose Stowaway [16], which provides a mapping of API calls to permissions and identifies over-privileged apps. It reports content providers that appear as single string literals, but does not examine other content provider API arguments or deal with string concatenation. Enck et al. propose ded [9] that decompiles Android dex code to get corresponding Java code. By using ded they provide insight into how Android apps behave. However, they don't provide much information about content provider use. There are several dynamic analysis tools for mobile apps. Enck et al. propose TaintDroid [17] that identifies that apps send privacy sensitive information to network servers using dynamic taint analysis. It found a number of potential information misuse from example apps. The underlying problem addressed by SYS is that Android users don't have enough knowledge or control over how apps use sensitive data. While SYS addresses this by providing more detailed information about app behavior, informing users decisions about whether to install an app, several recent works work dynamically to prevent apps from accessing sensitive data. TISSA [18], proposed by Zhou et al., and MockDroid [19], proposed by Beresford, supply fake information when there is a request for sensitive data from untrusted apps. AppGuard [20] modifies dex binaries so that security policies are checked and exceptions are thrown

when they are violated. Dynamic approaches could be useful for checking SYS's precision. On the other hand, SYS's analysis could potentially help in targeting the dynamic app of security policies.

V. CONCLUSION

In this paper we have described the threats that are imposed on Smartphone during installation. Our app will notify the user what the app is accessing on the Smartphone apart from the permissions that are asked from the users.

We have also added a functionality of malware detection. Hence we are aiming at making SYS a complete security for your android Smartphone.

VI. ACKNOWLEDGEMENT

I would like to convey my gratitude to Prof. Hasib Shaikh for giving me the constant source of inspiration and help in preparing the project, personally correcting my work and providing encouragement throughout the project. I also thank all my faculty members for steering me through the tough as well as easy phases of the project in a result oriented manner with concern attention.

REFERENCES

[1] N. Perlroth and N. Bilton, "Mobile apps take data without permission," NY Times, 2012. [Online]. Available: http://bits.blogs.nytimes.com/2012/02/15/google-and-mobile-appstake-data-books-without-permission/

[2] "Google play," https://play.google.com/store/apps.

[3] "Dedexer," http://dedexer.sourceforge.net/.

[4] "Android developer's guide," http://developer.android.com/guide/index.html.

[5] "Android sdk," http://developer.android.com/index.html.

[6] F. E. Allen and J. Cocke, "A program data flow analysis procedure," Commun. ACM, vol. 19, no. 3, pp. 137–, mar 1976.

[7] F. Tip, "A survey of program slicing techniques," JOURNAL OF PROGRAMMING LANGUAGES, vol. 3, pp. 121–189, 1995.

[8] "mobclix," http://www.mobclix.com/.

[9] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, "A study of android app security," in Proceedings of the 20th USENIX conference on Security, ser. SEC'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 21–21.

[10] M. Ongtang, K. Butler, and P. McDaniel, "Porscha: policy oriented secure content handling in android," in Proceedings of the 26th Annual Computer Security Apps Conference, ser. ACSAC '10. New York, NY, USA: ACM, 2010, pp. 221–230.

[11] W. Enck, "Defending users against smartphone apps: Techniques and future directions," in ICISS, 2011, pp. 49–70.

[12] A. Chaudhuri, "Language-based security on android," in Proceedings of the ACM SIGPLAN Fourth Workshop on Programming Languages and Analysis for Security, ser. PLAS '09. New York, NY, USA: ACM, 2009, pp. 1–7.

[13] A. P. Fuchs, A. Chaudhuri, and J. S. Foster, "Scandroid: Automated security certification of android apps."

[14] M. Egele, C. Kruegel, E. Kirda, and G. Vigna, "Pios: Detecting privacy leaks in ios apps," in Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, 6th February - 9th February 2011. The Internet Society, 2011.

[15] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, "Analyzing interapp communication in android," in Proceedings of the 9th international conference on Mobile systems, apps, and services, ser. MobiSys '11. New York, NY, USA: ACM, 2011, pp. 239–252.

[16] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in Proceedings of the 18th ACM conference on Computer and communications security, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 627–638.

[17] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," in Proceedings of the 9th USENIX conference on Operating systems design and implementation, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6.