Android Security via Static Program Analysis

Feng Shen
University at Buffalo, The State University of New York
{fengshen}@buffalo.edu

ABSTRACT

Android is a popular platform designed for mobile devices. It consists of a customized Linux kernel, middleware, and a few core applications such as the Phone application. The middleware, commonly referred to as the *Android framework*, provides libraries and runtime services to applications. Applications in Android are written mainly in Java. Once compiled, Android transforms its applications into the Dalvik Executable (or *DEX*) format to minimize the memory footprint. Android uses a Java VM called Dalvik to execute DEX bytecode. Unlike other mobile OSes, Android has a unique permission mechanism. At development time, an application developer needs to explicitly request permissions by including them in an application configuration file. We refer to this configuration file simply as the manifest in the remainder of the paper. At installation time, each user needs to review the permissions that the application requests and explicitly grant them.

Android currently has over 130 permissions applications can request in API level 17. These permissions are *API-oriented and access-based*, i.e., permissions control access to sensitive APIs (referred to as *protected APIs*). Generally, an application can ask for permissions to use protected APIs for phone resources (e.g, storage, NFC, WiFi, etc.) or information available on the phone (e.g., contacts, location, call logs, etc.). While this permission mechanism is effective in pinpointing which sensitive APIs that an application uses, it does not provide any insight into what the application actually does with the APIs. Thus, our goal is to complement the existing mechanism by providing both behavioral information of a single application as well as the interactions among multiple applications.

This thesis proposes Flow Permissions [3] [1], an extension to the Android permission mechanism. Unlike the existing permission mechanism, our permission mechanism contains semantic information based on information flows. Flow Permissions allow users to examine and grant per-app information flows within an application (*e.g.*, a permission for reading the phone number and sending it over the network) as well as cross-app information flows across multiple applications (*e.g.*, a permission for reading the phone number and sending it to another application already installed on the user's phone). Our goal with Flow Permissions is to provide visibility into the holistic behavior of the applications installed on a user's phone. In order to support Flow Permissions on Android, we have developed a static analysis engine, called BlueSeal, that detects flows within an Android application. Our system design is built on top of the Soot Java Optimization Framework [4]. We have also modified Android's existing permission mechanism and installation procedure to support Flow Permissions. Fig. 1 shows the procedure of our BlueSeal system architecture. At its core, our BlueSeal leverages classic forward and backward intraprocedural dataflow analysis as well as interprocedural dataflow analysis based on graph reachability. As outlined in Fig. 1, BlueSeal leverages six main analysis passes to generate Flow Permissions: 1) entry point discovery, 2) call graph restructuring, 3) unused permission analysis, 4) resolution of intents, content providers, as well as uses of the binder, 5) interprocedural permission flow analysis, and 6) cross-app permission flow analysis. Our evaluation compares our approach to dynamic flow tracking techniques; our results with 600 popular applications and 1,200 malicious applications show that our approach is practical and effective in deriving Flow Permissions statically.

Along with the explosive growth of smartphone sales, the threat of Android malware is spreading rapidly, especially those repackaged Android malware. This thesis proposes a new technique to detect mobile malware based on information flow analysis [2]. Our approach focuses on the *structure* of information flows we gather in our analysis, and the *patterns* of behavior present in information flows. Our analysis not only gathers *simple* flows that have a single source and a single sink, but also *Multi-Flows* that either start from a single source and flow to multiple sinks, or start from multiple sources and flow to a single sink. This analysis captures more complex behavior that both recent malware and recent benign applications exhibit. We leverage *N-gram analysis* to understand both unique and common behavioral patterns present in Multi-Flows. Our tool leverages N-gram analysis over sequences of API calls that occur along control flow paths in Multi-Flows to precisely analyze Multi-Flows with respect to app behavior. Using our approach, we show that there is a need to look beyond simple flows in order to effectively leverage information flow analysis for malware detection. By analyzing recently-collected malware, we show

MobiSys'17 PhD Forum, June 19, 2017, Niagara Falls, NY, USA

ACM ISBN 978-1-4503-4957-4/17/06.
DOI: http://dx.doi.org/10.1145/3086467.3086469

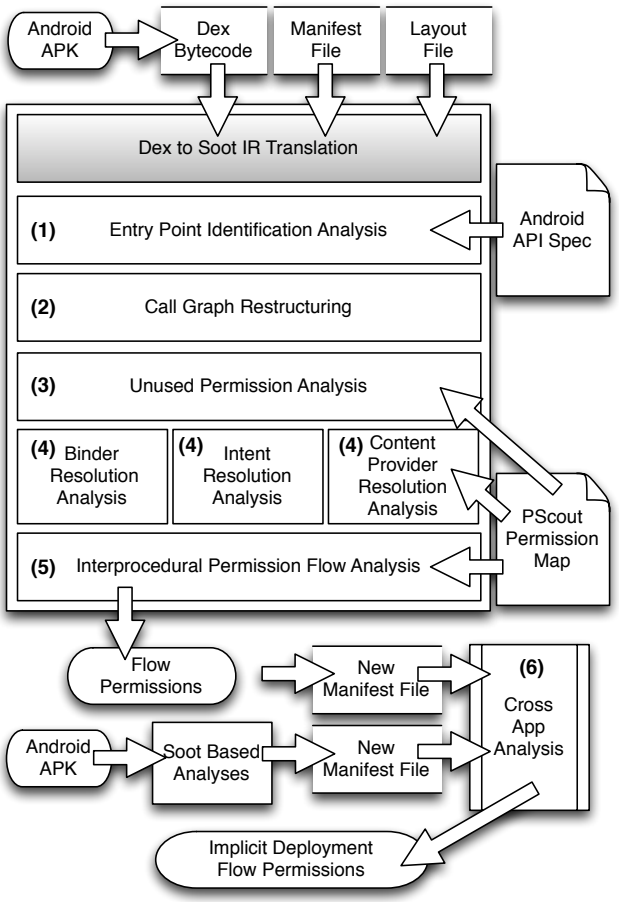

Figure 1: The BlueSeal Android app analysis framework architecture. Shaded boxes represent components already present in Soot.

there has been an evolution in malware beyond simply collecting sensitive information and immediately exposing it. Many previous systems focus on identifying the existence of *simple* information flows—i.e. considering an information flow as just a (source, sink) pair. However, modern malware performs complex computations before, during, and after collecting sensitive information and tends to aggregate data before exposing it. A simple (source, sink) view of information flow does not adequately capture such behavior.

The uniqueness of our approach comes from the following two features. First, our information flow analysis represent an information flow not as a simple (source, sink) pair, but as *a sequence of API calls*. This gives us the ability to distinguish different flows with same sources and sinks based on the *computation* performed along the information flow. Second, our information flow analysis detects Multi-Flows, flows that either start with a single source and flow to multiple sinks, or start with multiple sources and flow to a single sink. We treat such flows as a single flow, instead of multiple distinct flows. Fig. 2 shows a comparison of a Multi-Flow and its corresponding simple flows. This allows us to examine the *structure* of the flows themselves. We leverage machine learning techniques to extract features from Multi-Flows and their API sequences (N-gram analysis) and use these features to perform SVM-based classification. We show the precision of our technique by applying it to four different data sets totaling 8,598 apps. These data

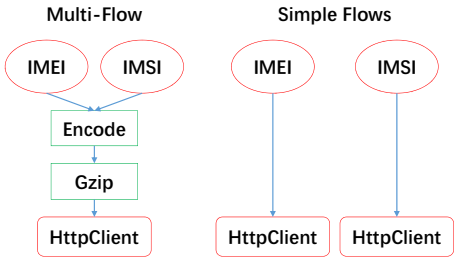

Figure 2: Multi-Flow vs Simple Flows

sets consist of both recent and older generation benign and malicious apps to demonstrate the effectiveness of our approach across different generations of apps.

Lastly, the Android market has been growing dramatically in the past decade. Along with it, both Android malware and benignware have been evolved and become more complicated. Due to the diverse functionalities modern apps provide, the benign apps are more complex and it is common for a benign app to leverage multiple sensitive data sources for normal usage. Besides, malware apps diguise themselves as benign apps and hide the malicious code among benign code. It becomes more and more difficult to distinguish malware apps from benign apps. In order to better understand modern malware apps, there is an urgent need to perform a systematical study on new Android malware.

This thesis proposes a systematical study on modern Android malware to characterize them from various aspects. More specifically, we will present a large collection of 1,000 modern Android malware apps, which will cover the majority of existing Android malware. We may also categorize these apps into different families based on the behavior similarity. We intend to release the whole dataset to the research community. Secondly, we will analyze these collected malware samples and thoroughly characterize them based on their detailed behavior from different aspects.

References

[1] S. Holavanalli, D. Manuel, V. Nanjundaswamy, B. Rosenberg, F. Shen, S. Y. Ko, and L. Ziarek. Flow permissions for android. In *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE 2013)*, 2013.

[2] F. Shen, J. Del Vecchio, A. Mohaisen, S. Y. Ko, and L. Ziarek. Android malware detection using complex-flows. In *Proceedings of The 37th IEEE International Conference on Distributed Computing Systems*, ICDCS '17.

[3] F. Shen, N. Vishnubhotla, C. Todarka, M. Arora, B. Dhandapani, E. J. Lehner, S. Y. Ko, and L. Ziarek. Information flows as a permission mechanism. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE '14.

[4] R. Vallée-Rai, P. Co, E. Gagnon, L. Hendren, P. Lam, and V. Sundaresan. Soot - a java bytecode optimization framework. In *Proceedings of the 1999 conference of the Centre for Advanced Studies on Collaborative research*, CASCON '99, pages 13–. IBM Press, 1999.