Jif-Based Verification of Information Flow Policies for Android Apps

Lina M. Jimenez, Systems and Computing Engineering Department, Universidad de los Andes, Bogotá, Colombia
Martin Ochoa, Singapore University of Technology and Design, Singapore
Sandra J. Rueda, Systems and Computing Engineering Department, Universidad de los Andes, Bogotá, Colombia

ABSTRACT

Android stores and users need mechanisms to evaluate whether their applications are secure or not. Although various previous works use data and control flow techniques to evaluate security features of Android applications, this paper extends those works by using Jif to verify compliance of information flow policies. To do so, the authors addressed some challenges that emerge in Android environments, like automatizing generation of Jif labels for Android applications, and defining translations for Java instructions that are not currently supported by the Jif compiler. Results show that a Jif-based analysis is faster and has a better recall than other available mechanisms, but it also has a slightly lower precision. Jif also provides an open source compiler, generates executable code for an application only if such application meets a defined policy, and checks implicit flows which may be relevant for highly sensitive applications.

KEYWORDS

Android, Information Flow Security Policies, Jif, Security-Typed Languages

1. INTRODUCTION

Controlling access and use of users’ data on Android platforms is key. First, Android is the most used platform in a market that keeps growing. Android had 78% of the market in 2013, and grew up to 80% in 2014 (Gartner Inc. 2015a). Furthermore, analysts project the number of mobile devices to grow to 2062 millions of units for 2017 (Gartner Inc. 2015b). Second, security in this platform worries experts as different evaluations have found multiple mobile applications that leak user’s data; while malware do it on purpose (Jiang and Zhou 2012), some applications do it by mistake (BBC 2012), (GSMA 2015). Finally, analysts warn about security in this context, as more people are using mobile devices for various tasks, ranging from management of personal data like family pictures and bank transactions, to management of data from their organizations, like official documents and e-mails (Symantec 2014), (Symantec 2015).

This work focuses on tools to support honest developers. While official stores can evaluate applications and remove those that do not meet expected behavior, it would be better for honest developers, that do not make errors on purpose, to have tools to check their applications before publishing them in stores and having them available to users. Errors in code may appear for several reasons, one of them being that developers cannot detect and fix errors on time because they do not normally use tools for security analysis as part of their development environments.

Traditional approaches to analyze program behavior use static and dynamic analyses. Tools based on static analysis use source code to build an abstract representation of a program’s behavior, and...
use that representation to evaluate particular properties, like reachability. Possible representations include control flow graphs, data flow graphs, and call dependency graphs (Bhosale 2014), (Fritz 2013), (Reps, Horwitz and Sagiv 1995). Tools based on dynamic analysis monitor a program behavior during execution, by using mechanisms like instrumentation, sandboxing, system calls traces, and dynamic taint marking (Clause, Li and Orso 2007), (Ernst, Static and dynamic analysis: synergy and duality 2003); if the tool detects a dangerous operation, it generates a warning and may even stop the operation if it has the required privileges. For both approaches programmers must first completely develop their applications, and only later they may use tools that are not part of their development environment to evaluate security of such applications. Adjusting applications after finishing them, if an evaluation finds errors, is also more expensive than evaluating and fixing them while they are under development.

This work proposes using Jif, a security-typed language, to build secure Android applications from early development phases, applications that would meet information flow security policies and avoid data leaks. Using a security typed language instead of the mentioned traditional approaches has several advantages: (1) security types enable developers to define various policies, according to particular requirements, (2) a security-typed language supports policy definition from the beginning of the development process, and (3) Jif only generates code that may be executed if the source code meets a defined policy. A developer may use this procedure to offer users a security guarantee; if the source code is available, an app store and even users, could verify policy compliance. Additionally, information flow policies (IFP) guarantee noninterference, that is high level input does not affect low level outputs, unless programmers explicitly use mechanisms to securely modify levels assigned to data (i.e. downgraders and endorsers).

Jif (Java + Information Flow) (Jif Cornell n.d.) is a Java extension that uses security types to enable developers define confidentiality and integrity information flow policies (IFP). This kind of policy defines how information may flow. For example, a simple confidentiality policy is: public → private; this policy specifies that public data may flow to private, but not the other way around. The Jif compiler allows developers to define this policy as part of the code, and generates executable code only if the source code meets the defined policy. However, although Jif handles Java code, and Android applications are written in Java, Jif cannot directly verify all Android applications.

This is not the first work that proposes using Jif to develop secure Android apps, but it is the first one that actually tries to use it and looks for options to overcome current Jif limitations. The contributions of this work are:

- It defines a confidentiality policy and assigns the corresponding Jif labels to Android resources, and Android API methods that enable data input and output. This assignment is required to analyze information flows of applications that use those resources and methods.
- It developed an automated Jif label generator to reduce the burden of manual label assignment on developers.
- The Jif compiler does not support all Java constructions. This work identifies two substitutions that enable automated analysis for applications that use the involved constructions. Although limited, this is a first step to be able to use Jif to compile Android apps.

The remaining sections are organized as follows: Section 2 presents an overview of Jif, Section 3 describes the developed mechanism to evaluate information flow policies for Android applications, based on the Jif compiler. Section 4 presents results, Section 5 synthesizes related work, and Section 6 concludes.
Real-Time Hardware-in-the-Loop in Railway: Simulations for Testing Control Software of Electromechanical Train Components
www.igi-global.com/chapter/real-time-hardware-loop-railway/66674?camid=4v1a

Extending Service-Driven Architectural Approaches to the Cloud
Raja Ramanathan (2013). Service-Driven Approaches to Architecture and Enterprise Integration (pp. 334-359).
www.igi-global.com/chapter/extending-service-driven-architectural-approaches/77955?camid=4v1a

Increasing the Accuracy of Software Fault Prediction using Majority Ranking Fuzzy Clustering
www.igi-global.com/article/increasing-the-accuracy-of-software-fault-prediction-using-majority-ranking-fuzzy-clustering/120519?camid=4v1a