Identifying Android Library Dependencies in the Presence of Code Obfuscation and Minimization

Salman A. Baset
IBM Research
sabaset@us.ibm.com

Shih-Wei Li
Columbia University
shihwei@cs.columbia.edu*

Philippe Suter
IBM Research
philippe.suter@gmail.com*

Omer Tripp
IBM Research
trippo@google.com*

I. INTRODUCTION

The Android platform has gained tremendous popularity since it was unveiled in 2007. Currently it holds a global market share of 78.4% with 1.5M activations of Android devices per day. In the US alone, there are 76M Android users. The main Android app market, Google Play, offers over 2M apps with a history of over 50B downloads to date. These impressive statistics provide strong motivation to understand, and improve the quality of Android apps. In this work, we focus on a key question in this space, which to our knowledge has not been previously addressed; namely:

*Can we determine the library dependencies of a given Android application without access to its source code?*

We emphasize that library dependencies include the exact library version, which is crucial information for many interesting use cases. First, an application may use a library (version) with known security vulnerabilities, or even worse, a library that is confirmed to be offensive if not malicious (e.g., an advertising library that make use of sensitive user information without proper authorization). Second, many libraries have known functional bugs, such as performance and memory problems [2]. Knowing the library dependencies of an application can help isolate bad behaviors exhibited by the application. Moreover, the ability to automatically extract the library dependencies of an application opens the door for large-scale studies on the use of libraries by Android apps, allowing insight into questions such as what the most popular libraries are, how their popularity changes across app categories, and how often a recent version of a library is used [4], [1].

Interestingly, though it may appear straightforward to compute the dependencies of an Android app without access to its source code, there are two serious challenges that complicate this task, sometimes to the point of rendering nonambiguous identification impossible. First, there is still strong incentive to reduce the size of Android apps (and mobile apps in general), also because many users still own legacy devices. Hence, through a process known as Minimization, Android apps typically do not include libraries in their entirety, but rather rely on reachability analysis to determine which parts of the library are used by the app (at the granularity of class members, i.e. fields and methods), such that only those class fragments are incorporated into the image, blurring the boundary between app and library code. Second, since mobile apps are downloaded onto end-user’s devices, the end-users can disassemble and inspect the app which might violate sensitive intellectual property and perhaps even uncover security weaknesses that render other users vulnerable. Thus, the app developers apply a process known as Obfuscation on the app, which not only obfuscates the source code written by developers but also to library symbols, since the boundary between app and library classes is blurred due to minimization. By the end of minimization and obfuscation, the app’s image, in the form of an .apk file, is largely a blob of obfuscated code with no immediate hints as to which libraries were incorporated into it. As an illustration, we refer the reader to Figure 1. Notice in particular the static call site highlighted in red with method identifier La/a/a/a/a/c/a. This is in fact a call to another method from the same Apache Commons Codec class (cf. the clear version in blue).

Our work presents a solution to the challenges highlighted above in the form of MOB SCANNER, a tool for identifying the precise versions of libraries used in the construction of an app without access to its source code. MOB SCANNER employs a combination of information-retrieval and constraint-solving techniques to report an effective set of candidate library dependencies (including their version), even when the app is obfuscated and/or when the set of potential libraries is very large.

II. TECHNIQUE

In this section, we explain the challenge in identifying the library dependencies of Android apps, namely because code obfuscation and minimization are by now an integral part of the Android build process. We then outline the main steps of our technique.

ProGuard and the Android Build Process. The Android build process packages an app into a single application package (.apk) file. The .apk file contains all the information necessary to run the app on either an emulator or a physical device. It includes compiled .dex files (Java .class files converted to
with relative ease. The first step is to compute a “signature” we assume the existence of a (comprehensive) database versions that an application is dependent on. Due to obfuscation, features correspond to elements of the code that are resistant those extracted from the classes of the libraries contained in D. Intuitively, the signature is a feature vector, where the features correspond to elements of the code that are resistant for each of the classes in A, and compare the signature against those extracted from the classes of the libraries contained in D. Intuitively, the signature is a feature vector, where the features correspond to elements of the code that are resistant.

We tackle these challenges via a unified approach that consists of two main steps. The input is an app A, where we assume the existence of a (comprehensive) database D of candidate library/version pairs. Repositories such as Bintray or Maven Central enable the construction of such a database with relative ease. The first step is to compute a “signature” for each of the classes in A, and compare the signature against those extracted from the classes of the libraries contained in D. Intuitively, the signature is a feature vector, where the features correspond to elements of the code that are resistant.

...
REFERENCES


