Poster: On Vulnerability Evolution in Android Apps

Jun Gao¹, Li Li², Pingfan Kong¹, Tegawendé F. Bissyandé¹, Jacques Klein¹

¹ Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg, Luxembourg
² Faculty of Information Technology, Monash University, Australia

ABSTRACT

In this work, we reconstruct a set of Android app lineages which each of them represents a sequence of app versions that are historically released for the same app. Then, based on these lineages, we empirically investigate the evolution of app vulnerabilities, which are revealed by well-known vulnerability scanners, and subsequently summarise various interesting findings that constitute a tangible knowledge to the community.

ACM Reference Format:
Jun Gao¹, Li Li², Pingfan Kong¹, Tegawendé F. Bissyandé¹, Jacques Klein¹

1 INTRODUCTION

Vulnerabilities of mobile apps, in general, and of Android apps, have been studied from various perspectives in the literature. Security researchers have indeed provided comprehensive analyses [1] of specific vulnerability types, establishing how they could be exploited and to what extent they are spread in markets at the time of study. The community has also contributed to improve the security of the Android ecosystem by developing security vulnerability finding tools [2] and by proposing improvements to current security models [3]. Unfortunately, whether these efforts have actually impacted the overall security of Android apps, remains an unanswered question. Along the same line of questions, little attention has been paid to the evolution of vulnerabilities in the Android ecosystem: which vulnerabilities developers have progressively learned to avoid? have there been trends in the vulnerability landscape? Answering these questions could allow the community to focus its efforts to build tools that are actually relevant for developers and market maintainers to make the mobile market safer for users.

Investigating the evolution of vulnerabilities in Android apps is however challenging. In the quasi totality of apps available in the marketplace, the history of development is a fleeting data stream: at a given time, only a single version of the app is available in the market; when the next updated version is uploaded, the past version is lost. A few works [4, 5] involving evolution studies have proposed to “watch” a small amount of apps for a period of time to collect history versions. Such studies are however biased towards excessively popular apps, which are written by highly-skilled and experienced teams. In this work, we set to perform a large scale investigation on how vulnerabilities evolve in Android apps.

2 EXPERIMENT SETUP

2.1 Terminology

An apk represents a released package of an app. All apks in our dataset are uniquely identified based on their hash. App version is used in our work to refer to a specific apk released in the course of development of an app. We then use both terms (apk and version) interchangeably. Finally, we define an app lineage as the consecutive series of its versions. In this work, a lineage may include only a subset of the apks that the app developers have released since our dataset, although massive, is not exhaustive.

2.2 App Dataset

AndroZoo currently hosts a dataset of over 5.5 millions distinct app packages (apks) from markets including the official Google Play store. According to its own description [6], apks are continuously collected to keep up with the evolution of apps in the Android ecosystem. API is provided to the community to download. We now describe the process (illustrated in Figure 1) which we have unfolded to re-construct app lineages from AndroZoo’s data heap. We consider four steps to (1) first conservatively identify unique apps, and (2) then link and order their app versions (i.e. apks) into a set of lineages. The objective is to maximize precision (i.e., a lineage will only contain apks which are actually different versions of the same app) even if recall may be penalized (i.e., not all apk versions might be included in a lineage). Indeed, missing a few versions will not threaten the validity of our study as much as linking together unrelated apps.

To avoid toy apps, we set a threshold of at least 10 apks before considering a lineage in our study. Overall, we were able to identify 28,564 lineages: the median size of the selected lineages is 13 apks and the total number of apks are 465,037.

2.3 Vulnerability Scanning

Vulnerabilities, also known as security-sensitive bugs, can be statically detected based on rules modeling vulnerable code patterns.
They are typically diverse in the components that are involved, the attack vector that is required for exploitation, etc. In this work, we focus on selecting common vulnerabilities with a severity level that justifies that they are highlighted in security reports and in previous software security studies. Before enumerating the vulnerabilities considered in our work, we describe the vulnerability scanning tools (a.k.a. scanners) that we rely upon to statically scan Android apps.

We stand on three state-of-the-art, open source and actively used scanners: FlowDroid, AndroBugs, and IC3.

- **FlowDroid** [7] – In the literature on Android, FlowDroid has imposed itself as a highly reputable framework for static taint analysis. It has been used in several works for tracking sensitive data flows which can be associated to private data leaks. The tool is still actively maintained.

- **AndroBugs** [8] was first presented at the BlackHat security conference, after which the tool was open sourced. This static scanner was successfully used to find vulnerabilities and other critical security issues in Android apps developed by several big players: it is notably credited in the security hall of fame of companies such as Facebook, eBay, Twitter, etc.

- **IC3** [9] is a state-of-the-art static analyzer focused on resolving the target values in *intent* message objects used for inter-component communication. The tool, which is maintained at Penn State University, can be used to track unauthorized Intent reception, Intent spoofing attacks, etc.

Table 1 summarises the vulnerability checks that we focus on, in accordance with the capabilities of selected scanners. Overall, we consider 10 vulnerability types. Although, the scanners report various alarms, we carefully selected those that represented vulnerabilities in app with a high level of criticality if exploited.

### REFERENCES


