Poster: On Vulnerability Evolution in Android Apps

Jun Gao[1], Li Li[2], Pingfan Kong[1], Tegawendé F. Bissyandé[1], Jacques Klein[1]
[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 Gao[1], Li Li[2], Pingfan Kong[1], Tegawendé F. Bissyandé[1], Jacques Klein[1]. 2018. Poster: On Vulnerability Evolution in Android Apps. In *ICSE '18 Companion: 40th International Conference on Software Engineering Companion, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3183440.3194968

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 fleeing 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 often 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.

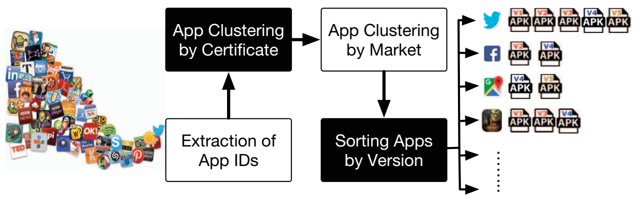

Figure 1: App Lineages Re-construction Process.

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.

Table 1: List of Considered Vulnerabilities.

Type	Vulnerability checking description	Scanner
	Security features	
SSL_Security	SSL Connection	AndroBugs
	SSL Certificate Verification	AndroBugs
	SSL Implementation (Verifying Host Name in Fields)	AndroBugs
	SSL Implementation (Verifying Host Name in Custom Classes)	AndroBugs
	SSL Implementation (WebViewClient for WebView)	AndroBugs
	SSL Implementation (Insecure component)	AndroBugs
Encryption	Base64 String Encryption	AndroBugs
KeyStore	KeyStore Protection	AndroBugs
	Permissions, privileges, sandbox, access-control	
Permission	App Sandbox Permission	AndroBugs
IntentFilter	Unauthorized Intent Reception	IC3
	Injection flaws	
Command	Runtime Command	AndroBugs
	Runtime Critical Command	AndroBugs
WebView	WebView RCE Vulnerability	AndroBugs
Fragment	Fragment Vulnerability	AndroBugs
	Data and Communication Handling	
Intent	Intent Spoofing	IC3
Leak	Sensitive Data Flow	FlowDroid

2.4 Methodology

Each of the vulnerability scanners outputs its analysis results in an ad-hoc format. We build dedicated parsers to automatically extract relevant information for our study. For the evolution study, we consider the analysis results for consecutive apk pairs in the lineages that we have re-constructed. Vulnerable pieces of code are extracted at the method and class levels following the locations indicated by the vulnerability scanners. These vulnerable pieces of code are collected and released as a valuable artefact for the community. Finally, we collect information on fix changes in a simple, although potentially coarse-grained, manner: given the analysis results for an apk v_1 and its successor v_2 in a lineage, we track the differences in terms of vulnerability locations; when a given vulnerability type is identified in a location but is no longer reported at the same location, we compute the change diff between the two apk versions and refer to it as *vulnerability fix changes.*

3 RESULTS

Our investigation into the vulnerability evolution in Android apps has revealed several interesting findings:

- Our analyses did not uncover any vulnerability bubble in the history of app markets. Instead, we note that vulnerabilities have always been widespread among apps and across time.
- Our large scale investigation of app lineages shows that apps do not get safer as they get updated: in general most vulnerabilities survive developer updates, and often times, new vulnerabilities are introduced by updated code.
- Vulnerability regressions occur in Android apps. This suggests an opportunity for the community to port regression testing techniques to address vulnerabilities detection during app updates.
- Third-party library code threatens the app security. This finding suggests that more focus should be given on the analysis of libraries towards a safer ecosystem. Market maintainers could draw policies rejecting apps using non-vetted libraries.
- Vulnerable apks could be updated into malicious versions later in the app lineage.

REFERENCES

[1] Li Li, Tegawendé F Bissyandé, Mike Papadakis, Siegfried Rasthofer, Alexandre Bartel, Damien Octeau, Jacques Klein, and Yves Le Traon. Static analysis of android apps: A systematic literature review. *Information and Software Technology*, 2017.

[2] Li Li, Alexandre Bartel, Tegawendé F Bissyandé, Jacques Klein, Yves Le Traon, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Damien Octeau, and Patrick McDaniel. Iccta: Detecting inter-component privacy leaks in android apps. In *Proc. of the 37th Intl. Conference on Software Engineering-Volume 1*, pages 280–291. IEEE Press, 2015.

[3] Meng Xu, Chengyu Song, Yang Ji, Ming-Wei Shih, Kangjie Lu, Cong Zheng, Ruian Duan, Yeongjin Jang, Byoungyoung Lee, Chenxiong Qian, et al. Toward engineering a secure android ecosystem: A survey of existing techniques. *ACM Computing Surveys (CSUR)*, 49(2):38, 2016.

[4] Vincent F Taylor and Ivan Martinovic. To update or not to update: Insights from a two-year study of android app evolution. In *Proc. of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 45–57. ACM, 2017.

[5] Li Li, Tegawendé F Bissyandé, Yves Le Traon, and Jacques Klein. Accessing inaccessible android apis: An empirical study. In *The 32nd Intl. Conference on Software Maintenance and Evolution (ICSME 2016)*, 2016.

[6] Li Li, Jun Gao, Médéric Hurier, Pingfan Kong, Tegawendé F Bissyandé, Alexandre Bartel, Jacques Klein, and Yves Le Traon. Androzoo++: Collecting millions of android apps and their metadata for the research community. *arXiv preprint arXiv:1709.05281*, 2017.

[7] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *Acm Sigplan Notices*, 49(6):259–269, 2014.

[8] Yu-Cheng Lin. Androbugs framework: An android application security vulnerability scanner. In *Blackhat Europe 2015*, 2015.

[9] Damien Octeau, Daniel Luchaup, Matthew Dering, Somesh Jha, and Patrick McDaniel. Composite constant propagation: Application to android inter-component communication analysis. In *Proc. of the 37th Intl. Conference on Software Engineering-Volume 1*, pages 77–88. IEEE Press, 2015.