Are Your Android App Analyzers Still Relevant?

Anonymous Author(s)

ABSTRACT

The diversity of mobile devices on the market fostered the emergence of cross-platform frameworks, the adoption of which can simplify the development and deployment of mobile applications on multiple platforms at once. Meanwhile, this trend also challenges the state-of-the-arts static program analysis techniques in terms of analyzing Android apps with soundness and completeness. To investigate the impact of cross-platform frameworks on static analyzers, we surveyed seven of the most popular cross-platform frameworks and proposed a tool in detecting the adoption of cross-platform frameworks. We also explored the prevalence of cross-platform frameworks in the most popular one hundred apps from Google Play and Tencent App Store. In addition, by investigating the cross-platform code and their location, we find that the state-of-the-arts Android static analyzers fail to analyze the cross-platform applications mainly because they lack the capability of handling Dart and JavaScript.

ACM Reference Format:
Anonymous Author(s). 2024. Are Your Android App Analyzers Still Relevant?. In *Proceedings of 11th International Conference on Mobile Software Engineering and Systems (MobileSoft 2024).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Android analysis analyzers have been used to analyze the bytecode of compiled Android apps through reverse engineering. Their primary goal is to assist developers, security experts, and researchers in identifying issues [1–3], vulnerabilities [4–6], malware [7–10], and potential risks [11–13] within applications. This, in turn, helps improve the quality, performance, and security of the applications [14, 15]. Among them, a large number of static analysis tools conduct in-depth analysis of an application's APK format (the released version as it is not always possible to access the application's source code) [5, 16, 17]. In such a case, they often need to first reverse engineer the application's bytecode and subsequently transform it into an intermediate representation, and finally conduct systematic code scrutiny to detect potential vulnerabilities and security risks, ultimately enhancing the security of applications.

In the mobile ecosystem, with over 2 million apps, Android has consistently maintained a leading position. At the same time, iOS also holds a significant market share, and yet there are new mobile platforms such as OpenHarmony [18] are about to emerge. This multi-platform situation challenges app providers maintain apps in an effective way. In fact, app providers have to keep multiple teams (e.g., one for Android, one for iOS, and possibly one for OpenHarmony) to keep their apps updated accordingly. To mitigate this challenge, our community has introduced sevral cross-platform development paradigms. By leveraging cross-platform frameworks, it is feasible to develop app once and deploy it everywhere across different mobile platforms.

Considering great benefits brought by cross-platform developments, we hypothesize (i.e., **Hypothesis 1**) that many apps (especially large and complex ones) are already to be developed with cross-platform frameworks. The immediate question following this phenomenon is whether existing Android app analyzers are still relevant. As mentioned earlier, the majority of static analyzers are proposed for directly analyzing Android APK files, which include DEX files that are eventually compiled from Java/Kotlin source code. In other words, the current form of analysis is explicitly tailored to DEX files. Unfortunately, Java/Kotlin is no longer the language of choice for cross-platform frameworks, which often utilize other languages such as Javascript to form mobile apps. We, therefore, hypothesize (i.e., **Hypothesis 2**) that those established static Android app analyzers are no longer effective for scrutinizing apps generated via a cross-platform paradigm.

In this work, we aim to understand the current status of cross-platform frameworks usages in the mobile market and subsequently validate the aforementioned two hypotheses through the following two research questions.

- **RQ1 (Hypothesis 1):** How are cross-platform apps spread in the current mobile ecosystem?
- **RQ2 (Hypothesis 2):** Can existing Android app analyzers be effectively applied to analyze cross-platform apps?

Our preliminary experimental results discover seven cross-platform frameworks, with React Native and Flutter being the most popular ones. The result further discloses that the distribution of cross-platform apps is quite different (over 60% of popular apps in China vs. around one-fifth worldwide). Moreover, as cross-platform Android apps have their core application code stored in other places rather than Dalvik, the majority of existing static app analyzers focusing on dissecting Dalvik bytecode become irrelevant when applied to cross-platform apps. There is hence a strong need for our community to invent dedicated approaches to analyze cross-platform apps.

2 EXPERIMENTAL SETUP

Dataset. To study the popularity of cross-platform applications in the market, we need to collect a set of real-world Android apps to fulfill our experiment exploitation. In this work, we resort to two markets (i.e., Google Play and Tencent App Store) to harvest Android apps. These two markets are selected to represent the app distribution outside and inside China, respectively. Google Play is the largest Android application market in the world, with the most complete applications and the largest number of users. Tencent App Store [19] is a third-party integrated management software for

MobileSoft 2024, April 2024, Lisbon, Portugal
© 2024 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Android smartphones launched by Tencent, and it is also one of the largest mobile phone application acquisition platforms in China. We downloaded the most popular 100 apps from Google Play and the Tencent App Store for our study.

Methodology. To answer the aforementioned two research questions, we first investigate the prevalence of cross-platform frameworks and then examine how these cross-platform frameworks have been integrated and implemented into Android apps in practise. To this end, given an Android app, we need to first reverse-engineer it and extract its app code. We achieve this by leveraging the famous Soot framework which can directly decompile Dalvik bytecode in Android APK to Jimple format, an intermediate representation that is easy to understand. Then, we need to go through the code to check if cross-platform frameworks are involved. In this work, we achieve this by extracting the code packages and comparing to the known cross-platform framework package names. We will then report our findings in the RQ1 at Section 3. Once cross-platform frameworks are identified, we go one step further (also manually) to understand how the code (written based on the cross-platform frameworks) copes with the Android execution mechanism. We will give more details in the RQ2 at Section 3.

3 RESULT

3.1 RQ1: How are cross-platform apps spread in the current mobile ecosystem?

Table 1 summarizes the list of cross-platform frameworks (i.e., the first column) used in the Top-100 apps from Google Play and Tencent App Store (the second and third columns), respectively. The second column of Table 1 presents the programming languages for cross-platform implementation in each cross-platform framework. In total, we have identified seven cross-platform frameworks, which are briefly introduced below.

Table 1: The list of identified cross-platform frameworks and their usages among popular Android apps.

		App Market	
Framework	**Language**	**Google Play**	**Tencent App Store**
Apache Cordova	JavaScript	1	4
Corona SDK	Lua	0	2
Flutter	Dart	2	39
Ionic Framework	JavaScript	1	0
React Native	JavaScript	14	30
Uniapp	JavaScript	0	1
Weex	JavaScript	0	14

(1) **React Native** is an open-source framework developed by Meta for simplifying cross-platform app development. React Native is based on the popular *React* framework [20], which is a Node.js-based JavaScript library used for creating web user interfaces. It allows developers to use a shared JavaScript codebase for both Android and iOS development encompassing Interface (UI) composition and other general business logic. Further, it features cross-language communication between JavaScript and the native side, blending native app performance with web development's flexibility and efficiency. Its flexibility allows it to build new apps from scratch or to be incorporated into the existing Android and iOS projects, with notable usage by *Facebook*, *Shopify*, and *Skype*.

(2) **Flutter** is introduced by Google aimed to quickly build high-quality mobile apps on iOS and Android[21]. Flutter introduced the programming language, Dart, as the developer's cross-language implementation. In the build and working process, Dart is compiled into binary code, and that's why it runs with the native performance of Objective-C, Swift, Java, or Kotlin. Both UI composition and business logic can be implemented through Dart. In addition, Flutter has integrated Hot-reload. This allows Flutter to automatically rebuild the UI widget tree, allowing users to quickly view the effects of changes. Numerous corporations use flutter, including *Uber*, *eBay*, *Alibaba*, etc[22].

(3) **Weex** is a framework for building high-performance cross-platform mobile applications with a modern web development experience, which enables developers to use modern web development skills to build Android, iOS, and Web apps with a single codebase[23]. Vue is a popular JavaScript framework for web apps with easy binding between data in memory and the user interface, and Weex allows users to code native mobile apps with the Vue framework. The language used by Weex is also JavaScript. Companies using Weex include *Alibaba*, etc.

(4) **Uniapp** is a framework for developing all front-end applications using Vue.js. Developers write a set of codes that can be published to iOS, Android, Responsive Web, and various mini programs[24]. Uniapp has a common front-end technology stack, which reduces the cost of learning. It also supports vue syntax and WeChat mini program API. The applications using Uniapp are mainly WeChat mini-programs.

(5) **Ionic** is an open-source UI toolkit for building performant, high-quality mobile apps using web technologies — HTML, CSS, and JavaScript — with integrations for popular frameworks like Angular, React, and Vue[25]. Ionic uses modern Web APIs such as Custom Elements and Shadow DOM, which have a stable API, and aren't at the whim of a single platform vendor. Companies using ionic include *Southwest Airline*, *Sanvello*, *H&R Block*, etc.

(6) **Apache Cordova** is an open-source mobile development framework. It allows users to use standard web technologies - HTML5, CSS3, and JavaScript for cross-platform development[26]. Compared with Ionic, Cordova focuses more on plugins. Plugins are an integral part of the Cordova ecosystem. Users can search for Cordova plugins using plugin search or npm, and develop their plugins. Businesses using Apache Cordova are *Walmart*, *Adobe*, *QrStore*, etc.

(7) **Corona SDK**, developed by Ansca, is an excellent option for any kind of mobile developer from beginner to advanced[27]. Corona uses the Lua programming language. Corona has Automatic OpenGL-ES Integration, so there is no need to call extensive classes or functions to create simple screen manipulations. The applications using Corona SDK include *Gojek*, *easypaisa - Payments Made Easy*, *Violin: Magical Bow*, and so on.

Observant readers may have noticed that, as also highlighted in Table 1, the distribution of cross-platform frameworks is quite different between Google Play and Tencent App Store. In Google Play, there are four cross-platform frameworks identified, and in total 17 apps (out of the top 100 apps) developed based on them. The most popular framework is React Native. In the Tencent App Store, there are six cross-platform frameworks leveraged, and in total 62 apps (out of the top 100 apps) developed based on them. Flutter is the most popular framework, followed by the React Native framework. The fact that over half of the apps are now developed via cross-platform apps in the Chinese market shows that the usage of cross-platform frameworks in China is significantly higher than that outside of China. Although we do not know the reason behind that, this result does motivate us to re-think if the existing static Android app analyzers proposed by our fellow researchers in the Mobile Software Engineering community are still relevant for dissecting the state-of-the-art Android apps. We hence go one step deeper to check that by manually exploiting the implementation mechanism of cross-platform frameworks when applied to generate Android apps. We will discuss our findings in the next subsection.

3.2 RQ2: Can existing Android app analyzers be effectively applied to analyze cross-platform apps?

To the best of our knowledge, existing app analyzers such as FlowDroid [11] or IccTA [28] are designed to analyze Android apps' Dalvik bytecode, which is compiled from Android platform-specific code (e.g., Java or Kotlin). During the development of cross-platform apps, the developer's implementation shifted from platform-specific code to other languages. Indeed, as highlighted in Table 1, none of the cross-platform frameworks take Java (or Kotlin) as their programming language. Since majority of static Android app analyzers focus on analyzing Java code, these approaches will not be able to directly analyze the source code project developed based on cross-platform frameworks.

However, it is not yet feasible to conclude that those existing analyzers cannot be applied to analyze Android apps built via cross-platform frameworks. Indeed, the build process provided by the cross-platform frameworks could still contain the platform-specific language like Dalvik bytecode (compatible with Java bytecode) to achieve native user experience. In such a case, existing analyzers should still be effective when applied to analyze cross-platform apps.

Towards confirming the aforementioned hypothesis, we go one step further to understand the build and working process of cross-platform apps (arrows in blue). Figure 1 illustrates an example drawn based on the popular React Native framework. To ease the understanding, and for comparison purposes, we also present the build and working process for native Android apps. As shown in Figure 1, for native Android apps, Java code will be compiled into Dalvik bytecode, which will be executed via a Dalvik Virtual Machine. The Dalvik code (same as the Java code) follows the Android's working paradigm, with components such as Activity to represent GUI pages and lifecycle methods to reflect the working mechanism of Android.

For React Native apps, the Javascript code (written by app developers) will not be compiled into Dalvik bytecode but put into a JS bundle. When building the app, the cross-platform framework will generate a Dalvik bytecode that contains wrapper code (following the Android's working paradigm) to achieve the native experience. The actual app functions are still within Javascript code that will be executed over a Javascript Engine. React Native supports three distinct JavaScript engines, namely Hermes, JavaScriptCore, and V8. These engines enable the execution of JavaScript code through the so-called *bridge* channel.

The aforementioned example clearly shows that the working process between native and cross-platform Android apps is different. The former one has the app's main code stored in Dalvik bytecode, while the latter one has its core code stored in other places in other formats. As a result, the majority of existing static Android app analyzers that are designed to analyze Dalvik bytecode are no longer relevant when applied to cross-platform Android apps. Considering that a significant number of apps are now developed via cross-platform frameworks (i.e., 62% top apps in China and around one-fifth of apps worldwide), we argue that our community should pay more attention to cross-platform apps and invent dedicated app analyzers to address the various issues of cross-platform apps.

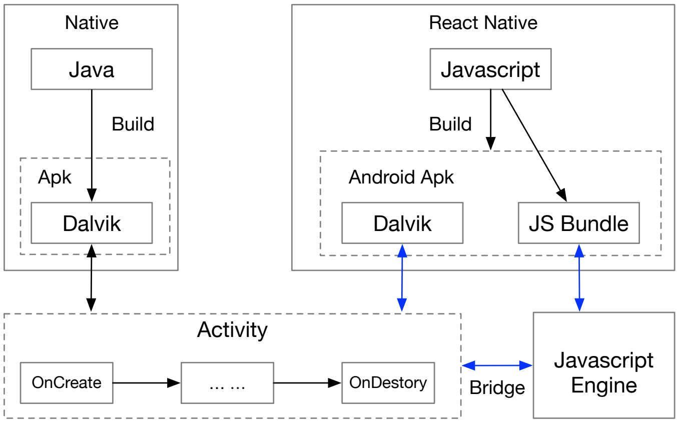

Figure 1: The build and working process of native Android app and React Native Android app.

4 CONCLUSION

We researched seven of the most popular cross-platform frameworks and developed a tool to determine their usage during app development. We conducted our experiments on the 100 most popular Android apps from the Google Play Store and Tencent App Store and analyzed them according to cross-platform framework types. The experimental results show that the proportion of usage of the cross-platform framework has reached more than 15% in the global market and even over 60% in the Chinese market. React Native and Flutter have the most users. Furthermore, we explored the language used by each cross-platform framework, along with the storage location and execution process of the corresponding code. This illustrates that today's Java-centric Android static analysis tools are unable to analyze those developers' cross-platform implementation in applications. Therefore, our community is in urgent need of static analysis tools developed specifically for applications built with these cross-platform frameworks.

REFERENCES

[1] L. Li, T. F. Bissyandé, H. Wang, and J. Klein, "Cid: Automating the detection of api-related compatibility issues in android apps," in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2018, pp. 153–163.

[2] L. Wei, Y. Liu, S.-C. Cheung, H. Huang, X. Lu, and X. Liu, "Understanding and detecting fragmentation-induced compatibility issues for android apps," *IEEE Transactions on Software Engineering*, vol. 46, no. 11, pp. 1176–1199, 2018.

[3] X. Sun, X. Chen, Y. Liu, J. Grundy, and L. Li, "Taming android fragmentation through lightweight crowdsourced testing," *IEEE Transactions on Software Engineering*, 2023.

[4] Y.-C. Lin, "Androbugs framework: An android application security vulnerability scanner," 2015, presented at Blackhat Europe 2015.

[5] Y. Liu, L. Li, P. Kong, X. Sun, and T. F. Bissyandé, "A first look at security risks of android tv apps," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*. IEEE, 2021, pp. 59–64.

[6] X. Sun, X. Chen, K. Liu, S. Wen, L. Li, and J. Grundy, "Characterizing sensor leaks in android apps," in *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2021, pp. 498–509.

[7] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, "Droidmat: Android malware detection through manifest and api calls tracing," in *2012 Seventh Asia joint conference on information security*. IEEE, 2012, pp. 62–69.

[8] S. Alam, Z. Qu, R. Riley, Y. Chen, and V. Rastogi, "Droidnative: Automating and optimizing detection of android native code malware variants," *computers & security*, vol. 65, pp. 230–246, 2017.

[9] X. Chen, C. Li, D. Wang, S. Wen, J. Zhang, S. Nepal, Y. Xiang, and K. Ren, "Android hiv: A study of repackaging malware for evading machine-learning detection," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 987–1001, 2019.

[10] Y. Liu, C. Tantithamthavorn, L. Li, and Y. Liu, "Deep learning for android malware defenses: a systematic literature review," *ACM Computing Surveys (CSUR)*, 2022.

[11] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," *Acm Sigplan Notices*, vol. 49, no. 6, pp. 259–269, 2014.

[12] L. Li, K. Allix, D. Li, A. Bartel, T. F. Bissyandé, and J. Klein, "Potential component leaks in android apps: An investigation into a new feature set for malware detection," in *2015 IEEE International Conference on Software Quality, Reliability and Security*. IEEE, 2015, pp. 195–200.

[13] Y. Liu, X. Chen, P. Liu, J. Grundy, C. Chen, and L. Li, "Reunify: A step towards whole program analysis for react native android apps," in *2023 IEEE/ACM International Conference on Automated Software Engineering*, 2023.

[14] Y. Hu, H. Wang, R. He, L. Li, G. Tyson, I. Castro, Y. Guo, L. Wu, and G. Xu, "Mobile app squatting," in *The Web Conference 2020 (WWW 2020)*, 2020.

[15] P. Liu, Y. Zhao, H. Cai, M. Fazzini, J. Grundy, and L. Li, "Automatically detecting api-induced compatibility issues in android apps: A comparative analysis (replicability studies)," in *The ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2022)*, 2022.

[16] L. Li, T. F. Bissyandé, M. Papadakis, S. Rasthofer, A. Bartel, D. Octeau, J. Klein, and L. Traon, "Static analysis of android apps: A systematic literature review," *Information and Software Technology*, vol. 88, pp. 67–95, 2017.

[17] J. Samhi, A. Bartel, T. F. Bissyandé, and J. Klein, "Raicc: Revealing atypical intercomponent communication in android apps," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1398–1409.

[18] L. Li, X. Gao, H. Sun, C. Hu, X. Sun, H. Wang, H. Cai, T. Su, X. Luo, T. F. Bissyandé *et al.*, "Software engineering for openharmony: A research roadmap," *arXiv preprint arXiv:2311.01311*, 2023.

[19] "Tencent app store," Online, July 10 2023, accessed: 2023-7-10. [Online]. Available: https://sj.qq.com/app

[20] *React*, 2023. [Online]. Available: https://react.dev/

[21] "Flutter - build apps for any screen," Online, 2023. [Online]. Available: https://flutter.dev/docs

[22] "Flutter apps in production," Online, 2023. [Online]. Available: https://flutter.dev/showcase

[23] "weex," Online, 2023. [Online]. Available: https://weexapp.com/zh/

[24] "uni-app," Online, 2023. [Online]. Available: https://uniapp.dcloud.net.cn/

[25] "The mobile sdk for the web." Online, 2023. [Online]. Available: https://ionicframework.com/

[26] "Apache cordova," Online, 2023. [Online]. Available: https://cordova.apache.org/

[27] "corona," Online, 2023. [Online]. Available: https://coronalabs.com/

[28] L. Li, A. Bartel, T. F. Bissyandé, J. Klein, Y. Le Traon, S. Arzt, S. Rasthofer, E. Bodden, D. Octeau, and P. McDaniel, "Iccta: Detecting inter-component privacy leaks in android apps," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1. IEEE, 2015, pp. 280–291.