A Proposed Framework for the Automated Authorization Testing of Mobile Applications

Ahmed Mohammed Alghamdi[1*] and Khalid Almarhabi [2],
amalghamdi@uj.edu.sa kamarhabi@uqu.edu.sa
[1]Department of Software Engineering, College of Computer Science and Engineering, University of Jeddah, Jeddah 21493, Saudi Arabia
[2]Department of Computer Science, College of Computing in Al-Qunfudah, Umm Al-Qura University, Makkah 24381, Saudi Arabia

Abstract
Recent studies have indicated that mobile markets harbor applications (apps) that are either malicious or vulnerable, compromising millions of devices. Some studies indicate that 96% of companies' employees have used at least one malicious app. Some app stores do not employ security quality attributes regarding authorization, which is the function of specifying access rights to access control resources. However, well-defined access control policies can prevent mobile apps from being malicious. The problem is that those who oversee app market sites lack the mechanisms necessary to assess mobile app security. Because thousands of apps are constantly being added to or updated on mobile app market sites, these security testing mechanisms must be automated. This paper, therefore, introduces a new mechanism for testing mobile app security, using white-box testing in a way that is compatible with Bring Your Own Device (BYOD) working environments. This framework will benefit end-users, organizations that oversee app markets, and employers who implement the BYOD trend.

Key words:
Authorization; BYOD; Mobile Applications, Testing

1. Introduction

The mobile applications (apps) marketplace is creating a paradigm shift in how software is delivered to end-users and enterprises. At the same time, the term Bring Your Own Device (BYOD) has become a widely-used technological trend, with more organizations allowing employees to use their personal mobile devices for work-related tasks. The mobile apps marketplace has many benefits, including the ability to quickly and effectively acquire, deliver, maintain, and improve software. However, the growth of this marketplace has increased security threats, which target the platforms that use mobile apps. These threats have, for instance, been seen in the Android apps marketplace, in which many apps have been infected by malware and spyware [1]. The Open Web Application Security Project (OWASP)—a nonprofit foundation that works to improve software security [2]—has published the OWASP Top 10, which is a standard awareness document for developers and web app security specialists, providing a broad consensus about the most critical security risks to web apps [3]. The first attack listed on the OWASP Top 10 is the accessing and publication of user information without user consent [2]. This attack occurs at the level of platform security controls in mobile devices' operating systems. However, some such risks can be prevented by ensuring good access control strategies. When a developer attempts to upload an app into an app store, the best practice is to check the app's access control policies and permissions. This step will ensure that apps meet the requirements for using permissions legally and properly.

This paper focuses on the Android platform because of its popularity, its status as an open-source platform, and, in general, the vulnerability of the app market. Access-control-related risks in mobile apps, which are also the focus of this research, may be identified as in Fig. 1. The types of informal access described in Fig. 1 must be prevented, and this research proposes a framework for detecting them. This framework will quickly, accurately, and automatically detect security vulnerabilities.

Security testing is a difficult task, especially as methods for exploiting apps are being continuously improved, and software and tools, requiring little effort and experience, have been developed to stage app attacks. Unlike functional testing, which tests the software itself to ensure that it meets specifications and works without errors, security testing is a form of negative testing; it ensures that the software can handle unexpected user behavior or invalid inputs. These invalid inputs or unexpected user permissions or behaviors will be fed into the framework developed in this research to reveal possible security vulnerabilities. Then, through negative testing, the framework will compare these vulnerability results

Manuscript received May 5, 2021
Manuscript revised May 20, 2021
https://doi.org/10.22937/IJCSNS.2021.21.5.30

with those obtained from the app's required permissions and expected user behaviors.

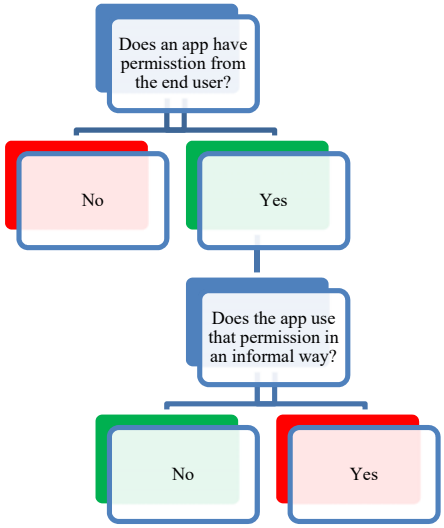

Fig 1: Risks regarding app permissions that must be addressed

2. Mobile Apps

A mobile app may be defined in many ways, but, simply, it is a software application or program designed to run on a mobile device (a smartphone or tablet), benefiting from its features and hardware components. There are three types of mobile apps: native, web, and hybrid [4], [5]. A native mobile app is developed exclusively for a single mobile operating system; that is, it is "native" to that specific platform or device. A web app functions as a website; it uses a web browser to run and is usually programmed via HTML, JavaScript, or CSS. A hybrid mobile app is a web app packed into a native app container, which combines the previously mentioned types.

According to the Android developer guidelines, each app has four main components: activities, services, broadcast receivers, and content providers [6]. Knowing the functions of each is important for understanding who can detect and prevent unusual behavior in an app. An activity is the entry point, represented by a single user-interface screen and consisting of several layouts. The service provides an entry point for keeping an app running in the background, performing long-running operations or processes. A broadcast receiver enables events outside of the regular user flow to be delivered to the app, which allows it to respond to the system broadcast or announcements messages, even if it is not currently running. The content provider manages shared app data, which can be stored in a file system on the web, in a database, or in any other storage location accessible to the app. Three of the four components (activities, services, and broadcast receivers) are activated by asynchronous intent messages. These intent messages bind individual components to each other at runtime, defining the actions to be performed for the activities and services and defining the announcement being broadcast. Activities and services for an app have lifecycles, as shown in Fig. 2 and described in [6], [7]. This diagram explains an app's status and functionality for detecting and preventing unusual behavior—a process that requires monitoring each app's files and formal permissions.

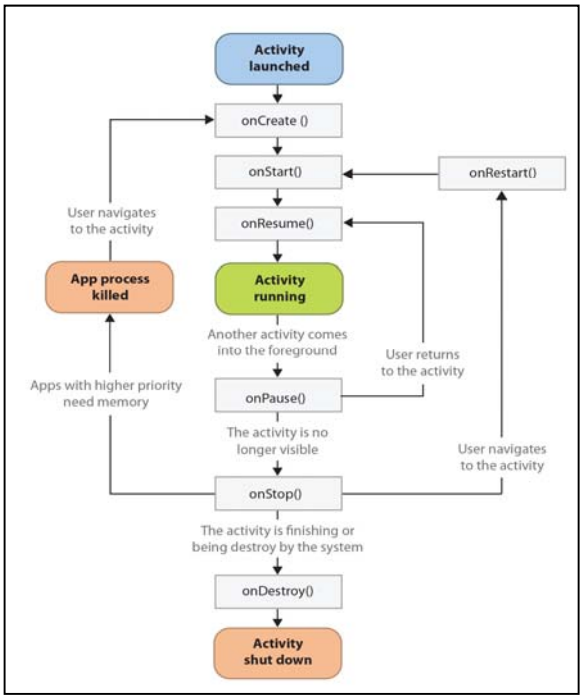

Fig 2: Application activity and service lifecycle in Android form [7]

3. Bring Your Own Device (BYOD)

There are various ways to achieve digital transformation, and one of the most suitable is the BYOD approach, which requires employees to use their personal devices (smartphones, tablets, USB drives, or computers) to connect to their organization's network and systems [8]. The BYOD concept was popularized after Cisco adopted it in 2009 [9], and its appeal has also been fueled by information technology (IT) consumerism. Employers largely permit employees to use their own mobile devices due to the advanced features devices for personal use often

possess. BYOD also has various advantages, such as reducing costs, boosting user productivity, and generating savings in procurement, software, hardware, service agreements, insurance, and licensing [10]. BYOD further enhances employee mobility, productivity, satisfaction, and flexibility.

When implementing BYOD, several points must be considered. First, BYOD boosts efficiency, as every employee is an expert at using his or her device, which minimizes the need for training [11]. Second, BYOD also helps companies provide services at minimum costs, even in rural areas. Third, employees who use their own devices are quite diligent in doing so [12], and, finally, information sharing and communication are instantaneous and can be accomplished from anywhere, even without local area network (LAN) or Wi-Fi availability [13]. However, for all its merits, increased BYOD implementation also increases the presence of users' extra-organizational apps, which, by being active on the users' personal devices, could affect companies' networks and infrastructures.

4 Related Work

The current authors' work related to this subject covers three categories: sandboxing, access control policies, and analysis of software behavior. Sandboxing is the security mechanism used to separate running programs, reducing system failures or preventing the spread of software vulnerabilities; sandboxing is used to execute untested or untrusted programs or codes from different sources, without risking the overall operating system or host machine [14], [15]. Android enforces app security by forcing each app to be executed in a secure Android sandbox. This sandbox is assigned a unique user identification (UID) for each Android app, and each sandbox runs its assigned app via its own process. In the device memory, each instance of one app is isolated from instances all other apps. To implement these sandboxes, Android uses various safeguards, which have been implemented and revised over time. The original UID-based discretionary access control (DAC) sandbox has been greatly expanded by these enforcements [16].

Access control is a set of rules that define which users have access to which services and what kinds of access restrictions are in place. When many operating systems, database management systems (DBMS), or network control systems use various access mechanisms, a user's ability to access system-protected resources may be improved. "Security aware applications" and "security ignorant applications" may be differentiated in any information system. Thus, finalized apps rely on control facilities given by an operating system, a DBMS, and some middleware. For example, the main access control issue is observed when suitable access control to data is applied in any information system. Reports are made frequently about poor access control management practices, which result in security and privacy violations [17].

Several studies have researched methods of detecting malicious software behaviors. Wang et al. [18] present a virus detection method focused on examining unusual behaviors revealed by application programming interface (API) sequences in Windows environments. This approach applies the Bayes algorithm to identify suspicious behaviors and detect viruses. Additionally, Beaucamps et al. [19] present a method for detecting malware that involves abstracting software behaviors. This technique abstracts software traces by rewriting original knowledge into abstract symbols that reflect their features. Suspicious, malicious behaviors are then detected by comparing trace abstractions. In terms of testing mobile apps, several studies have aimed to ensure mobile app quality via various techniques, including regression testing [20]; Mobile Application Testing (MAT), which involves functional, performance, and compatibility testing [21]; and the testing model published in [22]. An automated testing tool has also been designed to detect errors in Android apps on the cloud [7].

In summary, to the best of the current authors' knowledge, no previously published research compares app permissions, distinguishing between formal and informal access to specific app resources. However, this information is important for deciding whether an app is trusted or untrusted; untrusted apps should be excluded from mobile app markets.

5. Proposed Architecture

This section proposes a software architecture design for testing the authorizations of mobile apps—particularly Android apps (Fig. 3). This framework takes an Android Application Package (APK) file as input. Android apps are distributed and installed via APKs, which are Java bytecode packets. If the source code is not readily accessible, one of the available tools, such as Dedexer, is used to reverse engineer the

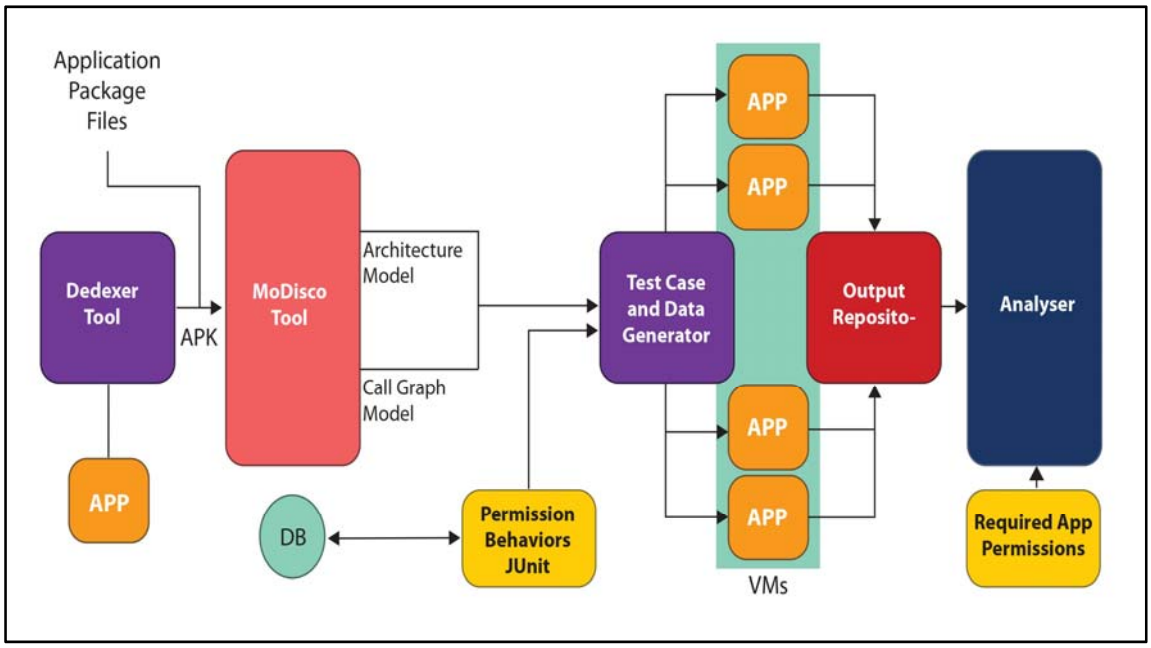

Fig. 3: Our Proposed Architecture

APK file. Then, a Call Graph Model and an Architectural Model are built via the MoDisco method.

These two models serve as the foundation for this research. The first error (the use of informal permissions) can be found via the Architectural Model, which represents an app's design and user interface configuration, made up of metadata for Android apps. The second error (the informal use of formal permissions) can be found via the Call Graph Model, which describes all possible method invocation sequences (execution traces) in an app.

As its name suggests, the Test Case and Data Generator generates test cases and data by combining the Architectural and Call Graph Models with all authorization activities stored in a database. A test case prototype is a skeleton Java file containing all the test case's common static elements. JUnit methods—for example, setUp() and tearDown()—are used as part of the template.

Testing will take place in four Android virtual machines (VMs) to increase the testing speed. All results will be collected from the VMs in an output repository. Finally, the analyzer model will determine whether the results are compatible with the required permissions and whether there are any bad behaviors in the Architectural and Call Graph Models.

5. Conclusion

Some app stores do not employ security quality attributes regarding authorization, which is the function of specifying access rights to access control resources. However, well-defined access control policies can prevent mobile apps from being malicious. This paper has introduced a new framework for determining app security detecting access control-related risks, which will benefit end-users, organizations overseeing app markets, and employers who implement the BYOD trend.

References

[1] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Doley et al., "Google Android: A Comprehensive Security Assessment," IEEE Secur. Priv. Mag., vol. 8, no. 2, pp. 35–44, Mar. 2010, doi: 10.1109/MSP.2010.2.

[2] OWASP, "The Open Web Application Security Project (OWASP)," 2020. [Online]. Available: https://owasp.org/about/.

[3] OWASP, "OWASP Top Ten," 2020. [Online]. Available: https://owasp.org/www-project-top-ten/.

[4] N. Serrano, J. Hernantes, and G. Gallardo, "Mobile Web Apps," IEEE Softw., vol. 30, no. 5, pp. 22–27, 2013, doi: 10.1109/MS.2013.111.

[5] S. Charkaoui, Z. Adraoui, and E. H. Benlahmar, "Cross-platform mobile development approaches," in 2014 Third IEEE International Colloquium in Information Science and Technology (CIST), 2014, pp. 188–191, doi: 10.1109/CIST.2014.7016616.

[6] Android Developers, "Android developer guides," Google, 2021. [Online]. Available: https://developer.android.com/docs.

[7] R. Mahmood, N. Esfahani, T. Kacem, N. Mirzaei, S. Malek et al., "A whitebox approach for automated security testing of Android applications on the cloud," in 2012 7th International Workshop on Automation of

Software Test (AST), 2012, pp. 22–28, doi: 10.1109/IWAST.2012.6228986.
[8] K. Almarhabi, K. Jambi, F. Eassa, and O. Batarfi, "Survey on access control and management issues in cloud and BYOD environment," Int. J. Comput. Sci. Mob. Comput., vol. 6, no. 12, pp. 44–54, 2017.
[9] A. B. Garba, J. Armarego, D. Murray, and W. Kenworthy, "Review of the information security and privacy challenges in Bring Your Own Device (BYOD) environments," J. Inf. Priv. Secur., vol. 11, no. 1, pp. 38–54, 2015, doi: 10.1080/15536548.2015.1010985.
[10] K. Almarhabi, K. Jambi, F. Eassa, and O. Batarfi, "An Evaluation of the Proposed Framework for Access Control in the Cloud and BYOD Environment," Int. J. Adv. Comput. Sci. Appl., vol. 18, no. 2, pp. 144–152, 2018, doi: 10.14569/IJACSA.2018.091026.
[11] M. Finneran, "Mobile security gaps abound," Information Week, 2012.
[12] P. K. Gajar, A. Ghosh, and S. Rai, "BRING YOUR OWN DEVICE (BYOD): SECURITY RISKS AND MITIGATING STRATEGIES," J. Glob. Res. Comput. Sci., vol. 4, no. 4, pp. 62–70, 2013.
[13] N. Zahadat, P. Blessner, T. Blackburn, and B. A. Olson, "BYOD security engineering: A framework and its analysis," Comput. Secur., vol. 55, pp. 81–99, 2015, doi: 10.1016/j.cose.2015.06.011.
[14] R. Wahbe, S. Lucco, T. E. Anderson, and S. L. Graham, "Efficient software-based fault isolation," in Proceedings of the fourteenth ACM symposium on Operating systems principles - SOSP '93, 1993, pp. 203–216, doi: 10.1145/168619.168635.
[15] V. Prevelakis and D. Spinellis, "Sandboxing Applications," in Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference, 2001, pp. 119–126.
[16] Android Developers, "Application Sandbox," Google, 2021. [Online]. Available: https://source.android.com/security/app-sandbox?hl=en.
[17] R. S. Sandhu and P. Samarati, "Access control: principle and practice," IEEE Commun. Mag., vol. 32, no. 9, pp. 40–48, Sep. 1994, doi: 10.1109/35.312842.
[18] C. Wang, J. Pang, R. Zhao, and X. Liu, "Using API Sequence and Bayes Algorithm to Detect Suspicious Behavior," in 2009 International Conference on Communication Software and Networks, 2009, pp. 544–548, doi: 10.1109/ICCSN.2009.60.
[19] P. Beaucamps, I. Gnaedig, and J.-Y. Marion, "Behavior Abstraction in Malware Analysis," 2010, pp. 168–182.
[20] Q. Do, G. Yang, M. Che, D. Hui, and J. Ridgeway, "Regression Test Selection for Android Applications," in 2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft), 2016, pp. 27–28, doi: 10.1109/MobileSoft.2016.023.
[21] C. M. Prathibhan, A. Malini, N. Venkatesh, and K. Sundarakantham, "An automated testing framework for testing Android mobile applications in the cloud," in 2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies, 2014, pp. 1216–1219, doi: 10.1109/ICACCCT.2014.7019292.
[22] B. N. Puspika, B. Hendradjaya, and W. Danar Sunindyo, "Towards an automated test sequence generation for mobile application using colored Petri Net," in 2015 International Conference on Electrical Engineering and Informatics (ICEEI), 2015, pp. 445–449, doi: 10.1109/ICEEI.2015.7352542.

Ahmed Mohammed Alghamdi is an assistant professor at the Software Engineering Department, College of Computer Science and Engineering, University of Jeddah, Saudi Arabia. He got his Ph.D. in Computer Science from King Abdulaziz University, Jeddah, Saudi Arabia. He received his B.Sc. degree in Computer Science from King Abdulaziz University, Jeddah, Saudi Arabia, in 2005 and the first M.Sc. degree in Business Administration from King Abdulaziz University, Jeddah, Saudi Arabia, in 2010. He received the second master's degree in Internet Computing and Network Security from Loughborough University, UK, in 2013. Dr. Ahmed also has over 11 years of working experience before attending the academic carrier. His research interests include high-performance computing, big data, distributed systems, programming models, software engineering, BYOD, and software testing.

Khalid Ali Almarhabi is an assistant professor at the Computer Science Department, College of Computing in Al-Qunfudah, Umm Al-Qura University, Saudi Arabia. He got his Ph.D. in Computer Science after studying this degree at both King Abdulaziz University, Jeddah, Saudi Arabia, and Queensland University of Technology, Brisbane, Australia. He also holds an MSc degree in Information Technology from Queensland University of Technology, Brisbane, Australia, in 2014. He holds a BSc degree in computer science from King Abdulaziz University, Jeddah, Saudi Arabia, in 2009. His research interests are information security, BYODs research, access control policies, information system management, and cloud computing.