Review Article

Survey on Reverse-Engineering Tools for Android Mobile Devices

Ashwag Albakri ¹, **Huda Fatima** ², **Maram Mohammed** ¹, **Aisha Ahmed** ¹,
Aisha Ali ¹, **Asala Ali** ¹, and **Nahla Mohammed Elzein** ³

¹Department of Computer Science, Jazan University, Jazan, Saudi Arabia

²Department of Information Technology & Security, Jazan University, Jazan, Saudi Arabia

³Faculty of Computer Sciences, The Futur University, Khartoum, Sudan

Correspondence should be addressed to Huda Fatima; hsaadullah@jazanu.edu.sa

Received 26 October 2021; Accepted 24 December 2021; Published 13 January 2022

Academic Editor: Shaojian Qu

Copyright © 2022 Ashwag Albakri et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

With the presence of the Internet and the frequent use of mobile devices to send several transactions that involve personal and sensitive information, it becomes of great importance to consider the security aspects of mobile devices. And with the increasing use of mobile applications that are utilized for several purposes such as healthcare or banking, those applications have become an easy and attractive target for attackers who want to get access to mobile devices and obtain users' sensitive information. Developing a secure application is very important; otherwise, attackers can easily exploit vulnerabilities in mobile applications which lead to serious security issues such as information leakage or injecting applications with malicious programs to access user data. In this paper, we survey the literature on application security on mobile devices, specifically mobile devices running on the Android platform, and exhibit security threats in the Android system. In addition, we study many reverse-engineering tools that are utilized to exploit vulnerabilities in applications. We demonstrate several reverse-engineering tools in terms of methodology, security holes that can be exploited, and how to use these tools to help in developing more secure applications.

1. Introduction

In the last decade, the development of information technology has been increasing dramatically which contributes to advancing the progress of society and the national economy. The Internet and its application have been an integral part of our life. It provides a convenient and fast way to obtain information and receive services remotely. In the way of information acquisition, the mobile devices represented by intelligent terminals are gradually replacing the traditional personal computers and are becoming a convenient channel for people to conduct network interconnection and obtain information anytime and anywhere [1].

The operating system is the core of intelligent devices. In the mainstream intelligent devices operating system, the Android system developed by Google has become the world's first share of the intelligent devices operating system in a few years with its open-source. According to the latest data released by the strategy analytics of market companies

[2], the global market share of Android phones reached a high record of 87.5% in the third quarter of 2016, which is in an absolute dominant position.

Due to the increasing share of Android intelligent operating systems and its open-source, the information related to the system vulnerabilities and malware exposure is common, which led to various security problems that have caused substantial harm to Android users. With the Android system, users can store files, send and receive e-mail, surf the Internet, do mobile payment, and log in to various applications such as banks, government, or health facilities. Sensitive information such as account numbers, passwords, bank card information, personal photos, and other private information are stored in Android mobile devices, which make hackers attempt to gain that valuable information of users easily by taking advantage of the lack of security of the Android system [3].

The application coded by developers may have program vulnerabilities due to a lack of secure coding techniques.

According to Tencent Mobile Security Labs [3], 9,182,500 new Android virus packages were found in the first half of 2018. In the first half of 2018, the number of mobile phone virus-infected users exceeded 200 million, a year-on-year increase of 42.35%.

Relatively the enormous growth of malware applications in the digital world has enforced the IT industries to deploy robust and high-quality applications to detect malicious content. Due to this, efficient products with signature-based intrusion detection features are designed in the market of the recent era due to its clear-cut adoptability. The ability to constantly update the signatures and restricted processing in the database has relatively been a reluctant approach in terms of IDS product updates. They also generate false negative alarms due to user overlapping behaviors. Therefore, alternate trending solutions are recommended to overcome the obstacles in the signature-based approach, such as cloud-based solutions. Such innovations can facilitate the antimalware corporations to switch to an automatic signature generation methodology, henceforth reducing false alarms [4].

In this research work, we aim at analyzing and reviewing the security threats and vulnerabilities in mobile devices running on the Android platform. We further investigate the several reverse-engineering tools and explain their methodology and how they can be used to exploit vulnerabilities in applications, so developers can utilize those tools to close those vulnerabilities and produce more secure applications.

2. Android System

The Android system has become one of the most popular mobile platforms for various devices. The Android system [5] is developed by Google which produces an average of two new releases per year that are embedded with an array of new features to improve the user interface and the system's performance [6]. Generally, the Android operating system is known for its mobility and is an open-source that is designed on the Linux kernel operating system [7].

2.1. Android System Architecture. The Android system is a layered operating system, and it can be divided into four levels, from bottom up: the primary layer being the Linux kernel, the secondary layer includes the Platform Libraries and the Android Runtime, the third layer is an application framework layer, and the top layer is applications layer [7], as shown in Figure 1.

2.1.1. Linux Kernel. Linux kernel is at the primary layer and is considered as the core of the Android architecture. Its primary role is to manage all the drivers of the device that play a vital role in the functionality of the Android device [8, 9].

2.1.2. Platform Libraries and Android Runtime. The secondary layer consists of components related to the libraries of C/C++ and various other scriptings to support the Android development like SSL, Graphics, and SQLite [10].

The core libraries play an important role in the Android runtime environment as it is the core application area that provides sufficient power to the Android applications and is considered as the foundational framework of the device.

2.1.3. Application Framework. The application framework is providing the system's functions to the applications. From the application developers' perspective, it is considered the most important layer because their work is mainly to develop applications that could run on top of the application framework layer and utilize the services provided by this application framework layer. More specifically, the framework layer provides access to all the APIs of the lower layer, to various hardware and software resources [11].

2.1.4. Applications Layer. The application layer interacts with users (i.e., applications (APP) in the form of .apk file) [12]. The applications are generally coded with the Java language that allows developers to code the program once and run it on multiple platforms. Along with applications developed by software developers, there are many system-level applications such as desktops, contacts, e-mail, and browsers [11]. The Android application mainly contains four components which are Activities, Services, Content Providers, and Broadcast receivers that provide a strong link among each other to build up the Android application.

3. Literature Review

There are multiple survey articles and academic papers publicized in recent years that focus on the security aspects of Android devices. In this section, we discuss several papers that survey and demonstrate the security of Android mobile devices.

Lima et al. [13] performed thorough research on securing and monitoring mobile device assets by discussing the mobile devices' security consequences and discussing the survey on the frameworks, innovations, and use cases for mobile device security management. Their research focused on studying the solutions of the management of the mobile devices, considering the preventive methods and techniques related to monitoring and mitigating threats at different nodes in the system. The purpose of this survey was to analyze the available security technologies that can be implemented to maximize the overall security of private data and applications stored and used by businesses and on Android devices.

Faruki et al. [14] focused on the Android platform and researched countless security concepts, such as code transformation processes, strength, and disadvantages of well-known malicious software analysis and techniques to detect. By studying different types of malicious software and various techniques used to solve various new types of malicious software, they resolved and concluded a wide-ranging evaluation framework that combines powerful inactive and active methods to be the solution to this new problem. In this paper, they discussed the implementation of the security of the Android device and its concerns and the

FIGURE 1: Android system architecture.

technological development of Android malware between 2010 and 2013, particularly malware penetrations, which hackers adopt in the backdoor procedure. This review provides an in-depth understanding of the advantages and disadvantages of known research methods, thereby providing a policy for scientists to propose future Android security, malware analysis, and malicious application detection methods.

Zachariah et al. [15] proposed virus detection methods for Android based on static analysis (i.e., permission-based detection, signature-based detection, and Dalvik bytecode detection) and discussed the merits and demerits of their methods. Furthermore, there was no extensive report and collectively 27 studies were included, with no discussion or presentation of future work conducted.

Demissie et al. [16] used static and dynamic taint analysis with the primary aim of detecting secured-based vulnerabilities. Considerably, taint analysis usually extracts data dependencies among request data from other applications and data used in primary operations. As a basic function in Android, sender requesting an operation from another application may not always cause a weakness. To reduce false alarms, a specific analysis method should differentiate between expected permission and actual permission re-authorization flows.

Felt et al. [17] introduced the authority of the redelegation problem in the Android operating system. Their approach identifies every expected access point of an application and analyzes the flow of data from the access point till the API is reached. They acknowledged certain procedures that can detect permission redelegation situations but cannot identify intended cases and vulnerable cases, henceforth intimating several false alarms.

Guyton [18] described the threats faced by mobile devices running Android in terms of malware types and many other well-known attacks, such as Trojan, BotNet, Backdoor, Worm, Spyware, Ransomware, and Aggressive Adware. The authors used four concepts like Support Vector Machines (SVMs), Self-Organizing Maps (SOMs), decision trees, and Bayesian networks. Those methods can be utilized for malware detection analysis that aims at combating Android malware.

Zou et al. [19] introduced a summary of the technical probabilities for virtual or nonvirtual attacks, jeopardizing privacy, taking control over communication, and malicious activities. This study explores the mechanisms involved in providing security threats and its environment, wherein a summarized form of attacks occurs over the Internet and intranet like compromising privacy, taking control over users' privileges, and so on. Moreover, investigations were done on how the malicious activity can be detected and how effective defense methodologies can be reinforced at the competitive Android development platform.

Ahmed and Sallow [20] discussed the present Android security threats and the present security proposed solutions and attempted to classify the proposed solutions and evaluate them. Their proposed solution is divided into two groups such as static and dynamic satisfying three goals: assessment, analysis, and detection. This review paper aimed for expanding the coverage of malicious application growth and Android security threats.

Shabtai et al. [21] identified the high-risk threats to the framework and proposed several security solutions to mitigate the threats. Their assignment proposed several security countermeasures, and based on these proposed countermeasures, the following recommendations can be

shortlisted: initially, a mechanism should be implemented that can prevent an attack at the primary layer of the kernel layer specifically by the SELinux access control mechanism. Constant vulnerabilities and limitations have gained much access on Android-based devices. Later on, the application needs stronger concepts to protect the permission mechanism or for the detection of any attacks. Consequently, the topmost priority is given to SELinux and other security mechanisms like firewalls and antivirus applications.

Bhat and Dutta [22] classified intrusions on the applications of Android into major groups: hardware abstraction layer-based attacks, hardware-based attacks, Linux-Shell-based attacks, and application-based attacks. Their research focused on a variety of threats and the procedures that can be taken in terms of security within these groups. They have also analyzed and summarized the most interrupting problems in Android applications. A comparative study was also made between different methodologies used in detecting various malicious activities, and their weaknesses were highlighted too.

Sikder et al. [23] provided a better spectrum of Android's performance at the system level, their policy issues, and guidelines for the engineers for taking strict measures while framing the rules and protocols in developing applications. The generation development of the Androids operating system was also studied and how the users should maintain their privacy to avoid the risks involved in maintaining a secured environment. Their study identified guidelines, recommendations about the safety, and security of smartphone platforms. They concluded that the companies producing smartphones must form a unified protocol to make sure all patches are updated based on trending developments. In conclusion, authorized practices should be practiced at large scales and uniformity must be maintained among software engineers. After all, proper development practices should be spread and made available to the developers by the platform maintainers.

4. Reverse-Engineering Tools

In this section, we explore twelve of the common reverse-engineering tools in terms of tool's methods, vulnerabilities a tool can exploit, and recommendations for developers.

4.1. APKInspector. The main purpose of this tool is to help in visualizing compiled Android packages and their corresponding DEX code. It provides a platform for analysis functions and graphic features to keenly observe the malicious activity within the apps. APKInspector is useful for displaying graphs of the Control Flow of a method and also allows in dragging and zooming. Also, it helps in displaying the codes of Java apps, shows their bytecodes, and displays classes, methods, and strings. It is also useful in tracing the User Interface dialogue before the malicious API. It can detect vulnerabilities like Insecure Data Storage, External file access, Weak Server-Side Controls, and World-writable files.

4.2. Apktool. Apktool is used to decompile APK into Java source code (i.e., classes.dex convert to jar file); after decompiling, all sources files of APP (i.e., XML file, Java files, AndroidManifest.xml, pictures, and others source files) will be viewed. Apktool is used for reinstalling the resources to originality that includes all the functions. Also, it allows initializing decoded segments into binary form. It manages APKs and Smali debugging and solves all repeated tasks.

The major benefit of Apktool in comparison with other tools is that it is two-way and user-friendly. Developers can break the code of a segment and transform it and then rebuild again using the available tools. This will result in a perfect code of segment and will create a new .apk file.

Apktool can help to detect many kinds of vulnerabilities like Untrusted Content-Type Header, Potential Command Injection, World-writable file (Android), and Potential Path Traversal using Scala API.

4.3. FindSecurityBugs. This tool is an extended version for *FindBugs* that includes a protocol for securing Java applications. The plugin can be integrated with various development tools. It can be used within Eclipse, IntelliJ, and Netbeans with their respective FindBugs plugin. Also, it is used for improving the efficiency of detectors that are static and discovering vulnerabilities and implements detectors based on availability. It also enhances documentation procedures for new contributors.

For developers, FindSecurityBugs helps and enables the developer to change and adapt to any new potential security threats. Developers can design a new customized security detector and new detection rules in FindSecurityBugs to increase the security vulnerability check coverage. It helps developers of Java web applications and Android applications for security audits. FindSecurityBugs can detect 128 vulnerability variants that include Command Injection, XPath Injection, SQL/HQL Injection, XXE, and Cryptography limitations.

4.4. AndroBugs Framework. It is a highly demanded framework used by developers or penetration testers with the ability to uniquely scan and detect vulnerabilities in Android devices. It is based on python language development, no installation, and configuration-related environment can be used. Although AndroBugs Framework does not provide GUI, it has high efficiency and accurate problem location (the average scanning time is less than 2 minutes).

AndroBugs Framework is useful for detecting vulnerabilities in applications of Android devices that include software exploits by hackers, checking if the code falls short of any efficiency, and allows checking susceptible shell commands. It collects useful data from high-impact APPs and checks the application's security protection.

AndroBugs Framework can help to detect vulnerabilities like SQL/HQL Injection, World-writable file (Android), Insecure Data Storage, and Untrusted Content-Type Header.

4.5. Bytecode Viewer. It is an open-source written in Java platform and is used for scanning for malware activities with the Malicious Code Scanner plugin. It easily edits APKs via Smali/Baksmali integration and securely launches Java servlets and inserts hooks via EZ-Injection. It can detect vulnerabilities like Untrusted Content-Type Header, Potential code injection, and Activities that may leak data.

4.6. JADX. It uses command-based tools and also consists of graphical interactive tools to give output in Java through the apk files. JADX tool is simple to install and used for uncompiling bytecode to Java servlets from files of APK, dex, and zip basis [24]. It helps in breaking the code of AndroidManifest.xml and other programs. It can detect vulnerabilities like Untrusted Content-Type Header, Potential code injection, Insecure Data Storage, and Untrusted Content-Type Header.

4.7. Super. This tool is used in operating systems like Windows, Macintosh, and Linux which can run command-based tools within the .apk files to look for weaknesses.

It can build security-oriented mobile applications where the code can be constantly reviewed and directed in HTML report that enable better vulnerability analysis.

Super can detect several vulnerability types like XPath Injection, SQL/HQL Injection, Potential LDAP Injection, and Potential code injection when using Script Engine.

4.8. AndBug. It helps in the software bugs used on virtual machines and has been a support for software analyzers and troubleshooting teams. It makes use of the similar interface of Android's Eclipse debugging plugin, Java Debug Wire Protocol (JDWP), and Dalvik Debug Monitor (DDM) to permit users to make use of Dalvik methods, investigation status, and also make alternatives. It is useful to act as an API tracer (for java classes) and is used to understand what is happening when a specific class/method is loaded when the APP is running. It can deal with several vulnerability types including Potential code injection, Insecure Data Storage, and Untrusted Content-Type Header.

4.9. Quick Android Review Kit (Qark). Many innovated applications are more vulnerable at the source code or packaged APKs; therefore, this kit allows to focus on the security issue of Android applications. It also can create evidence of any vulnerability that has been exploited.

Quick Android Review Kit (Qark) quickly and fastly detects common vulnerable issues within an Android device. It gives proper documentation for the reviewers of existing vulnerabilities within the application. It can use more than one compiler to have a wide spectrum of outputs that helps in refined results. It can detect several types of security vulnerabilities like eavesdropping and data leakage.

4.10. Introspsy-Android. While applications are running, this tool allows us to understand the process run time of the

Android device and helps in identifying the security issues. Following are the components of the Introspsy-Android, namely, an interface to apply hooks and filters and the ability to analyze issues.

Introspsy-Android is used for understanding the process of the application by the tester, and it can identify dangerous behaviors. It can detect security vulnerabilities like potentially dangerous behaviors, activities that may leak data, Enabled Backup Mode, and Potential code injection.

4.11. Inspeckage. Any Android application must be analyzed in a dynamic environment by the application of hooks to the various Android APIs. It allows a deep understanding of the application during the processing time. Inspeckage is used in vulnerabilities and malware analysis, and it has hook templates that show what the APP does during runtime.

Inspeckage is a technique that lets the programmer inject their code before and after activity, enabling a change in the system and application behavior. It helps in detecting vulnerabilities like data exposure, unauthenticated requests within the network, components within the Android device that have been transferred, and backup option activated and allows the application bugging.

4.12. Drozer. Drozer [25] has the ability for users to search for hidden vulnerabilities in applications and devices by focusing on the performance of the application and also interacts with the virtual machines of Dalvik and other crucial endpoints of the applications. It is one of the most widely used security frameworks implemented in Androids Pentesting applications. The built-in tools allow developers to exploit it as a multitasking application. It also can remotely exploit the device through various social engineering kits. It helps detect vulnerabilities like data that contain sensitive and attack-prone text, outdated server applications, an outcast of data unintentionally, and poor authorization and authentication.

Table 1 summarizes the vulnerabilities that can be exploited by each of the aforementioned reverse-engineering tools.

5. Security Countermeasures

In this section, we provide several security countermeasures [26] that users and developers can follow to enhance the security of the applications installed in Android mobile devices [27] and how malware can be detected on Android devices [28].

For users,

- (1) It is a good practice to regularly update the phone's software which includes security patches, features, and functionalities
- (2) It is good to avoid installing applications from untrustworthy sources [29, 30]
- (3) It is advisable to apply data protection using a password or keys to access sensitive data stored on mobile devices [31]

TABLE 1: Summary of the reverse-engineering tools.

Tool	SQL/ HQL Injection	Untrusted Content- Type Header	Weak Server- Side Controls	Insecure Data Storage System	World- writable file	Potential path API	Potential Command Injection	External file access	XPath Injection	Command Injection
APKInspector			✓	✓	✓			✓		
Apktool		✓			✓	✓	✓			
FindSecurityBugs	✓			✓					✓	✓
AndroBugs	✓	✓		✓	✓					
Framework										
Bytecode viewer		✓					✓			✓
JADX		✓		✓			✓			
SUPER	✓					✓	✓		✓	
AndBug		✓		✓			✓			
Quick Android						✓			✓	✓
Review Kit (Qark)						✓				✓
Introspey-Android	✓		✓			✓				✓
Inspeckage			✓	✓				✓		
Drozer				✓	✓		✓			

- (4) Two-step verification should be used to enhance the security of your device
- (5) Backups should be utilized to protect against data losses [32, 33]

For developers,

- (1) Security mechanisms should be utilized to secure the servers and API to prevent any form of unauthorized access
- (2) Since the data in mobiles are highly confidential, encryption procedures should be timely applied [34, 35].
- (3) Protecting the source code must be prioritized at all levels of analysis and compilers to secure and enhance the confidentiality of the intellectual property
- (4) A two-factor authentication system should be applied to protect users' sensitive data [36, 37]

6. Conclusion

This paper explains security concepts and the analysis of the vulnerabilities in Google's Android system with its open-source in the last few years, introducing in detail the structures of the Android system, analyzing and reviewing the security threats and vulnerabilities in mobile devices running on the Android platform. In this research work, we survey the literature on the application security of Android mobile devices. We demonstrate many reverse-engineering tools in terms of the tool's methodology, its uses, and vulnerabilities that can be.

We demonstrate many reverse-engineering tools in terms of the tool's methodology, its uses, and vulnerabilities that can be exploited. With reverse-engineering tools, developers can innovate highly secured applications. We will work towards designing an effective vulnerability detection tool for mobile applications, which we consider as future work.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

- [1] K. Kavitha, P. Salini, and V. Ilamathy, "Exploring the malicious android applications and reducing risk using static analysis," in *Proceedings of the 2016 International Conference on Electrical, Electronics, and Optimization Techniques*, pp. 1316–1319, ICEEOT, Chennai, India, March-2016.
- [2] Android, "Android security 2015 year in review," 2018, https://source.android.com/security/reports/Google_Android_Security_2018_Report_Final.pdf.
- [3] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, pp. 627–638, Chicago, USA., Oct-2011.
- [4] W. Enck, D. Ocateau, P. D. McDaniel, and S. Chaudhuri, "A study of android application security," in *Proceedings of the 20th USENIX Security Symposium*, vol. 2, p. 2, San Francisco, Aug-2011.
- [5] P. Kaur and S. Sharma, "Google android a mobile platform: a review," in *Proceedings of the 2014 Recent Advances in Engineering and Computational Sciences*, pp. 1–5, RAECS, Chandigarh, India, Mar-2014.
- [6] M. Haris, B. Jadoon, M. Yousaf, and F. H. Khan, "Evolution of android operating system: a review," *Asia Pacific Journal of Contemporary Education and Communication Technology*, vol. 4, no. 1, pp. 178–188, 2018.
- [7] elinux, "Android architecture," vol. 6, p. 13, 2014.
- [8] M. Jones, "Anatomy of the linux kernel," 2007, <https://developer.ibm.com/technologies/linux/articles/l-linux-kernel/>.
- [9] Dalvik, "Technical information," 2017, <http://source.Android.com/tech/dalvik/index.html>.
- [10] Google Inc, "Android – an open handset alliance project," 2007, <http://developer.android.com>.
- [11] I. Almomani and M. Alenezi, "Android application security scanning process," in *Telecommunication Systems-Principles and Applications of Wireless-Optical Technologies* IntechOpen, London, UK., 2019.

- [12] P. Barry and P. Crowley, *Modern Embedded Computing: Designing Connected, Pervasive, Media-Rich Systems*, Elsevier, Amsterdam, Netherlands, 2012.
- [13] A. Lima, B. Sousa, T. Cruz, and P. Simoes, "Security for mobile device assets: a survey," in *Proceedings of the ICMLG2017 5th International Conference on Management Leadership and Governance*, p. 227, Academic Conferences and publishing limited, Braamfontein, Johannesburg, 2000, South Africa, Mar-2017.
- [14] P. Faruki, A. Bharmal, V. Laxmi et al., "Android security: a survey of issues, malware penetration, and defenses," *IEEE Commun. Surv. tutorials*, vol. 17, no. 2, pp. 998–1022, 2014.
- [15] R. Zachariah, K. Akash, M. S. Yousef, and A. M. Chacko, "Android malware detection a survey," in *Proceedings of the 2017 IEEE International Conference On Circuits and Systems*, pp. 238–244, ICCS, Thiruvananthapuram, Kerala, India, Dec-2017.
- [16] B. F. Demissie, D. Ghio, M. Ceccato, and A. Avancini, "Identifying android inter app communication vulnerabilities using static and dynamic analysis," in *Proceedings of the International Conference on Mobile Software Engineering and Systems*, pp. 255–266, Austin, TX, May-2016.
- [17] A. P. Felt, H. J. Wang, A. Moshchuk, S. Hanna, and E. Chin, "Permission re-delegation: attacks and defenses," in *Proceedings of the 20th USENIX Security Symposium*, vol. 30, p. 88, San Francisco, August-2011.
- [18] F. Guyton, "A survey of android security threats and machine learning techniques used for detection," , 2018.
- [19] F. Zou, S. Zhang, T. Wan, and L. Pan, "A survey of android mobile platform security," in *Proceedings of the 10th International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 520–527, WiCOM, Larnaca, Cyprus, Oct-2014.
- [20] O. Ahmed and A. B. Sallow, "Android security: a review," *Acad. Journal Nawroz University*, vol. 6, no. 3, pp. 135–140, 2017.
- [21] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer, "Google android: a comprehensive security assessment," *IEEE Security Privacy*, vol. 8, no. 2, pp. 35–44, 2010.
- [22] P. Bhat and K. Dutta, "A survey on various threats and current state of security in android platform," *ACM Computing Surveys*, vol. 52, no. 1, pp. 1–35, 2019.
- [23] R. Sikder, S. Khan, S. Hossain, and W. Z. Khan, "A survey on android security: development and deployment hindrance and best practices," *Telkomnika*, vol. 18, no. 1, pp. 485–499, 2020.
- [24] A. B. Georgiev, A. Sillitti, and G. Succi, "Open source mobile virtual machines: an energy assessment of dalvik vs. art," in *Proceedings of the IFIP International Conference on Open Source Systems*, pp. 93–102, San José, Costa Rica, May-2014.
- [25] L. Xiaopeng, W. Ning, X. Fei, Q. Fengchen, and M. Simin, "Safety detection method of android app based on drozer," in *Proceedings of the 2018 International Conference on Smart Grid and Electrical Automation*, pp. 170–172, ICSGEA, Changsha, China, Jun-2018.
- [26] S. Bier, B. Fajardo, O. Ezeadum, G. Guzman, K. Z. Sultana, and V. Anu, "Mitigating remote code execution vulnerabilities: a study on tomcat and android security updates," in *Proceedings of the 2021 IEEE International IOT, Electronics and Mechatronics Conference*, pp. 1–6, IEMTRONICS, Toronto, ON, Canada, May-2021.
- [27] T. Zhao, G. Zhang, and L. Zhang, "An overview of mobile devices security issues and countermeasures," in *Proceedings of the 2014 International Conference on Wireless Communication and Sensor Network*, pp. 439–443, Wuhan, China, Dec-2014.
- [28] W. Yuan, Y. Jiang, H. Li, and M. Cai, "A lightweight on-device detection method for android malware," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 51, no. 9, pp. 5600–5611, 2019.
- [29] Y. Shen and G. Stringhini, "ANDRUSPEX: leveraging graph representation learning to predict harmful app installations on mobile devices," 2021, <https://arxiv.org/abs/2102.06132arXiv Prepr. arXiv2103.05476>.
- [30] S. Arasavalli, Y. Sravya, S. Venuturumilli, P. Tottempudi, and G. Ramakoteswarrao, "Securing android devices from snooping apps," in *Proceedings of the 2018 2nd International Conference on Trends in Electronics and Informatics*, pp. 1–5, ICOEI, Tirunelveli, India, May-2018.
- [31] I. M. Almomani and A. Al Khayer, "A comprehensive analysis of the android permissions system," *IEEE Access*, vol. 8, Article ID 216671, 2020.
- [32] S. Hong, C. Liu, B. Ren, Y. Huang, and J. Chen, "Personal privacy protection framework based on hidden technology for smartphones," *IEEE Access*, vol. 5, pp. 6515–6526, 2017.
- [33] J. Han and S. Lee, "A practical approach to analyze smartphone backup data as a digital evidence," in *Proceedings of the DFRWS USA*, Seattle, WA, USA, August-2016.
- [34] X. Lu, Z. Pan, and H. Xian, "An efficient and secure data sharing scheme for mobile devices in cloud computing," *Journal of Cloud Computing*, vol. 9, no. 1, pp. 1–13, 2020.
- [35] S. B. Olaleye and S. Kant, "Secure use of cloud storage of data on smartphones using atomic aes on arm architectures," *International Journal of Applied Engineering Research*, vol. 13, no. 5, pp. 2569–2581, 2018.
- [36] A. Kellner, M. Horlboge, K. Rieck, and C. Wressnegger, "False sense of security: a study on the effectivity of jailbreak detection in banking apps," in *Proceedings of the 2019 IEEE European Symposium on Security and Privacy*, pp. 1–14, EuroS&P, Stockholm, Sweden, Jun-2019.
- [37] G. Ali, M. Ally Dida, and A. Elikana Sam, "Two-factor authentication scheme for mobile money: a review of threat models and countermeasures," *Future Internet*, vol. 12, no. 10, p. 160, 2020.