Journal of Computer Science and Technology Studies
ISSN: 2709-104X
DOI: 10.32996/jcsts
Journal Homepage: www.al-kindipublisher.com/index.php/jcsts

JCSTS
AL-KINDI CENTER FOR RESEARCH AND DEVELOPMENT

| RESEARCH ARTICLE

Security Analysis of Android Applications a Case Study of Applications in Palestine.

Deema Sawalha[1], Saleh Salous[2], and Mahmoud Sawalha[3] ✉
[1]*Computer Systems Engineering, Palestinian Technical University-Kadoorie, Tulkarm, Palestine*
[2]*Computer Science, Palestinian Technical University-Kadoorie, Tulkarm, Palestine*
[3]*Communications Engineering Technology, Palestinian Technical University-Kadoorie, Tulkarm, Palestine*
Corresponding Author: Author's Name, Mahmoud Sawalha, **E-mail**: mahmoud.sawalha@ptuk.edu.ps

| ABSTRACT
The smart phone is becoming the most used computing and communication device. Users and service providers tend to use mobile phone applications as a means for using and providing different services. Android mobile operating system is the most prevalent among all mobile platforms. In Palestine telecommunication companies where the first to adopt the use of smartphone applications to provide services that include the transaction of billing information, location information, contacts and multiple private information about their customers. This presents the issue of how these applications are protecting and preserving the personal information of the users. This paper aims to analyze security vulnerabilities for selected mobile applications in Palestine, and the impact of these vulnerabilities on confidentiality, integrity, and availability of the user's data. The tools used in this paper are Mobile Security Framework (MobSF) and Quick Android Review Kit (QARK). This analysis can help to take appropriate actions so that the impact of these vulnerabilities can be reduced.

| KEYWORDS
Android Operating System; Security Vulnerabilities; Mobile Security Framework; Quick Android Review Kit.

| ARTICLE INFORMATION
ACCEPTED: 20 May 2025 **PUBLISHED:** 11 June 2025 **DOI:** 10.32996/jcsts.2025.7.6.7

1. Introduction

The prevalence of mobile applications has significantly increased over the last decade, with the extensive use of these applications in nearly all areas of daily activities including gaming, learning, shopping, among others. If a mobile's application has security vulnerabilities, it may compromise the user's security and privacy and may cause significant effects on the user's quality of life. This issue highlights the importance of proper security review and analysis of mobile applications.

Android is the most prevailing operating system for mobile devices. Security analysis for Android applications is a critical process that helps identify and assess security vulnerabilities in these applications. The process includes testing the application's security features to ensure that they are working as intended (*Android Developers*, n.d.-a)[1]. The Android built-in security framework uses a sand-box for the application that has a unique Linux user ID to protect it from accessing other applications' data stored in the same device (Dar, 2017) [2]. However, the way these applications handle the user data and the permissions required by these applications may cause potential risks to user's security.

The applications selected are for the major telecommunication companies in Palestine. The applications are made anonymous to preserve the privacy of the applications creators and users. The applications are assigned as app A, app B and app C. The APK's files of these applications where sourced from the Google Play Store and were extracted for testing using the security analysis tools.

The objective of this paper is to present a preview of the security measures taken by application creators with a local Palestinian application as a case study example. The three applications selected as a sample for the smartphone applications created for the Palestinian customers. The application security parameters are analyzed to determine the measures taken to preserve the integrity, confidentiality, and availability of user's data. This paper is organized as follows: Section 2 presents a literature review, Section 3 presents the methodology, Section 4 presents results, Section 5 presents discussion and analysis, and Section 6 presents conclusion and future work.

2. Literature Review

The authors of (Dar, 2017)[2] suggested a system aimed at raising user awareness by applying innovative security mechanisms and techniques to monitor apps behavior when trying to access sensitive resources and notifying the user during runtime. In (Sarkar et al., 2019)[3] the authors highlighted the challenges posed by evolving platforms and the advancements in security mechanisms regarding permissions, secure storage, authentication and encryption and the challenges posed by evolving platforms.

This article (R. Li et al., 2022) [4] offered a comprehensive security evaluation of Android custom permissions. It investigated the effectiveness of custom permissions in enhancing the security of Android applications. In (Yadav et al., 2017)[5] the authors discussed common vulnerabilities observed in Android apps, such as privilege escalation and code injection. While in (Meng et al., 2018)[6], the researchers built an Android vulnerability detection tools and focused on finding security issues caused by misuse of intents during app development. The authors of(D. Li et al., 2018) [7] discussed how to use deep learning for Android malware detection. Furthermore, (Khadiranaikar et al., 2017)[8] The authors proposed efficient approach to detect permission leakage vulnerabilities in Android Inter-process communications. In (Gong et al., 2020)[9] The authors highlighted the importance of code quality analysis in identifying potential vulnerabilities. In (Nirumand et al., 2020) [10], a proposal of a heavyweight extension of the Knowledge Discovery Metamodel (KDM) for modeling Android security was introduced.

The authors in (Şahin et al., 2018) [11] focused on permission-based static analysis techniques for Android malware detection. By analyzing the permissions pattern, the researchers aimed to distinguish between benign and malicious applications. In (Wu et al., 2017) [12], the researchers presented a practical static analysis approach involving analyzing the control-flow and dataflow for detecting intent-based permission leakage in Android applications to identify potential security risks. In (Rahman et al., 2017) [13] The authors used static code metrics as predictors to evaluate the risks through the development process.

3. Methodology

There are several techniques used in security analysis for Android applications, including automated tools and manual testing. Automated tools can help identify common vulnerabilities such as text traffic, SQL injection, and buffer overflow attacks. Manual testing, on the other hand, involves using a combination of techniques in order to simulate some attacks, then find security weakness points in a mobile application, and acquire access to sensitive data.

In this study, automated security tools were used to test the security in three of the applications used in Palestine, referring to them by A, B, and C, randomly. The APKs of these applications obtained for analysis using two security tools: the Mobile Security Framework (MobSF) and the Quick Android Review Kit (QARK).

The Mobile Security Framework (MobSF) is a web-based, automated, mobile app pen-testing, provides malware analysis, and performs security evaluation using a framework capable of performing static and dynamic analysis (*MobSF*, n.d.)[14]. This tool supports mobile app binaries (APK, XAPK, IPA, and APPX). For the purpose of this study, the APK file for each application was uploaded to the MobSF tool for static analysis.

A MobSF report provides a full analysis of possible security vulnerabilities in an Android application (*MobSF Manual*, n.d.)[15]. The report includes: Static Code Analysis using MobSF which is a tool built to detect probable security vulnerabilities and areas of concern for Android apps. The findings from the security analysis are categorized into several key areas presented in the results section of this article.

The second tool used for application security analysis is QARK (Quick Android Review Kit), which is a tool designed to determine potential security vulnerabilities in Android applications (*Android Developers*, n.d.-b; *Qark*, n.d.)[16][17]. It can analyze both source code and packaged APKs. A QARK report offers a comprehensive analysis of potential security vulnerabilities in an Android application(*Qark*, n.d.) [16]. The APK files of the applications were analyzed by executing QARK commands on Python. An HTML report file was generated for each application, detailing the detected vulnerabilities and specifying whether the issue was considered safe, warning, or vulnerable.

4. Results

After conducting the analysis, each application in this study was evaluated using two separate reports generated by the MobSF and QARK tools.

4.1 Application "A"

The APK file for Application 'A' was uploaded for static analysis, and a report was generated indicating the potential vulnerabilities detected by both the MobSF and QARK tools.

MobSF Report:

This report assigned Application 'A' a grade of B, with a security score of 47/100, indicating that the app poses a medium risk.

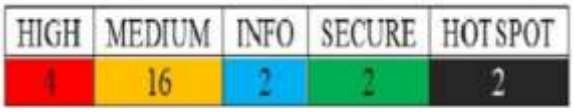

Figure.1: Application "A" Findings Severity.

Figure.1 illustrates the severity of the findings for application 'A', with four high-risk findings in the report's sections: certificate analysis, network security, MANIFEST analysis, and code analysis, respectively. The high-risk issues are shown in Table.1.

Issue	Severity	Description
Certificate algorithm vulnerable to hash collision	High	Possible hash collision caused by using SHA1.
5 domains of Application "A"	High	Insecure domain configuration by using clear text traffic
Enabled Clear text traffic for App	High	Explicit use of cleartext network traffic causes reduced security performance that includes authenticity and confidentiality, with a possibility of data tampering during transmission.
Insecure Implementation of WebView, ignoring SSL Certificate errors and accepting any SSL Certificate.	High	Inappropriate Certificate Validation that causes possible vulnerability of Man-In-The-Middle attacks.

Table.1: High Severity Findings for application "A".

The report identified several application permissions as dangerous, as shown in Table.2. The potential security risks include access to file location, unencrypted text transmission and several permissions that include camera and microphone access violating personal data. In addition to permissions that allow the application to access personal contacts and pictures information. Thes permissions are reported as dangerous by the MobSF tool which means there could be significant threats to user privacy and data security.

QARK Report:

Upon running the QARK static analysis on Application 'A', the report highlighted several issues under the warning of potential vulnerability. Some other issues were reported as dangerous vulnerabilities.

The first issue was the use of the RSA encryption algorithm. Several encryption and security values where set to default values. These default values could potentially allow unauthorized access to the application's code. The second issue, is that private keys are embedded directly in the application's code or configuration file. An attacker can exploit it if they gain access to the application's code. The third issue is 'Empty pending intent', which is related to the use of Pending Intents. These intents allow an application to perform a predefined code in this application's context. If a Pending Intent is created with an empty intent, it can be intercepted, redirected, or modified by malicious applications.

The QARK tool detected several application permissions as potential security threats. These include 'WebView' threat, which enables file and content access; 'detected logs' threat, which allow leakage of information from the Android application; 'insecure functions' threat, which provides no permission checking on the Content Provider API, thereby potentially allowing the API key to be obtained; and 'write to external storage' threat, which grants access to application's files to write and read their contents.

PERMISSION	STATUS	INFO	DESCRIPTION
'ACCESS_COARSE_LOCATION'	dangerous	coarse location (network based)	Access coarse location sources, to determine approximate phone location.
'ACCESS_FINE_LOCATION'	dangerous	fine location (GPS)	Access fine location sources, as Global Positioning System on phones. May consume extra battery power.
'RECORD_AUDIO'	dangerous	recording audio	Allow the access of audio record path.
'READ_EXTERNAL_STORAGE'	dangerous	reading external storage contents	Allow reading external storage.
'WRITE_EXTERNAL_STORAGE'	dangerous	reading/deleting/ modifying external storage contents	Allow writing to external storage.
'READ_CONTACTS'	dangerous	reading contact data	Allow reading of all contact data stored on phone. Suspicious applications can send this data to other people.
'CAMERA'	dangerous	taking pictures and videos	Allow taking pictures and videos with camera, which allows collecting images that is seen by the camera at any time.
'READ_PHONE_STATE'	dangerous	reading phone state and identity	Allow the access of device's phone features. With accessing this permission; phone number and its serial number can be determined, as well the connected number if a call is active.

Table.2: Dangerous Application "A" Permissions

A 'broadcast without receiver check permission' threat was also found in the QARK report. This means that any application on the device can receive this broadcast since the messages are broadcast without specifying the recipients. Another issue in the app was the 'Backup allowed in manifest', which allows data theft through local attacks if the device has USB debugging enabled. Similarly, 'Phone number or IMEI detected' issue means the app can access the phone number or the International Mobile Equipment Identity (IMEI) of the device. The issue 'Encryption keys are packaged with application' indicates that the app is embedding encryption keys within the app, which could be exploited.

4.2 Application "B"

The APK file for Application 'B' was uploaded to perform static analysis, and a report was generated indicating the possible vulnerabilities detected by both MobSF and QARK tools.

MobSF Report:

HIGH	MEDIUM	INFO	SECURE	HOTSPOT
1	17	3	1	2

Figure.2: Application "B" Findings Severity.

The MobSF report assigned Application 'B' a grade of B, with a security score of 51/100, indicating that the application poses a medium risk. Figure.2 shows the severity of the findings for application 'B' with one high risk finding in the report's section MANIFEST analysis. The issues with high risk shown in Table.3.

Issue	Severity	Description
Enabled Clear text traffic for App	High	Explicit use of cleartext network traffic causes reduced security performance that includes authenticity and confidentiality.

Table.3: High Severity Findings for application "B".

The report identified several application permissions as dangerous, as shown in Table.4.

QARK Report:

The QARK report for application 'B' identified several issues as threats and categorized them as dangerous. For example, the 'Empty pending intent' issue which is related to the use of Pending Intents in several application's files.

PERMISSION	STATUS	INFO	DESCRIPTION
'READ_EXTERNAL_STORAGE'	dangerous	reading external storage contents	Allow reading external storage.
'WRITE_EXTERNAL_STORAGE'	dangerous	reading/deleting/ modifying external storage contents	Allow writing to external storage.
'READ_PHONE_STATE'	dangerous	reading phone state and identity	Allow the access of device's phone features. With accessing this permission; phone number and its serial number can be determined, as well the connected number if a call is active.
'RECORD_AUDIO'	dangerous	recording audio	Allow the access of audio record path.
'SYSTEM_ALERT_WINDOW'	dangerous	displaying system-level alerts	Allows to show system-alert windows. This can be used to take over the screen of the phone.
'ACCESS_COARSE_LOCATION'	dangerous	coarse location (network-based)	Access coarse location sources, to determine approximate phone location.
'CAMERA'	dangerous	taking pictures and videos	Allow taking pictures and videos with camera, which allows collecting images that is seen by the camera at any time.

Table.4: Dangerous Application "B" Permissions

The tool also detected other application's permissions and categorized them as a warning of potential security threats. These warnings include 'detected logs', 'write to external storage', 'broadcast without receiver check permission', 'Phone number or IMEI detected', 'Backup allowed in manifest'. The 'vulnerable check permission function' indicates that a particular permission has been granted to some application. The 'vulnerable check permission function' threat indicates that a particular permission has been granted to some application. It can cause a privilege escalation attack if a malicious application gains access to this application's privileges by bypassing its permission check, and the 'insecure functions' provide no permission checking on the Content Provider API, so API key can be obtained. These issues could lead to information leakage, privilege escalation attacks, and data theft.

4.3 Application "C"

The APK file for Application 'C' was uploaded for static analysis, and a report was generated indicating the possible vulnerabilities detected by both MobSF and QARK tools.

MobSF Report:

HIGH	MEDIUM	INFO	SECURE	HOTSPOT
3	22	2	0	1

Figure.3: Application "C" Findings Severity.

This report assigned Application 'C' a grade of B, with a security score of 43/100, indicating that the app poses a medium risk. Figure.3 illustrates the severity of the findings for application 'C', with three high-risk findings in the report's sections Code analysis and Trackers. The issues with high risk are shown in Table.5.

Issue	Severity	Description
Insecure WebView Implementation, accepting any SSL Certificate and ignoring errors	High	Inappropriate Certificate Validation that causes possible vulnerability of Man-In-The-Middle attacks.
Enabled Remote WebView debugging	High	Mobile Applications Weakness due to Inappropriate Platform Usage.
Privacy Trackers	High	Trackers analytics include third-party components to track behavior or collection of user data.

Table.5: High Severity Findings for application "C".

The report identified several application permissions as dangerous, as shown in Table.6.

PERMISSION	STATUS	DESCRIPTION
'ACCESS_FINE_LOCATION'	dangerous	Fine location (GPS)
'ACCESS_COARSE_LOCATION'	dangerous	Coarse (network-based) location
'READ_CONTACTS'	dangerous	Reading contact data
'WRITE_CONTACTS'	dangerous	Writing and modifying contact data
'RECORD_AUDIO'	dangerous	Recording audio
'WRITE_EXTERNAL_STORAGE'	dangerous	Reading/deleting/modifying external storage contents
'READ_EXTERNAL_STORAGE'	dangerous	Reading external storage contents
'CAMERA'	dangerous	Taking pictures and videos
'READ_PHONE_STATE'	dangerous	Reading phone state and identity

Table.6: Dangerous Application "C" Permissions.

QARK Report:

The QARK report for application 'C' identified several issues considered as threats and categorized them as Warning, and Dangerous. The issue recognized as a dangerous vulnerability is 'Empty pending intent'. Some of the permissions recognized as potential security threats in the category 'Warning' are: detected logs, insecure functions, external storage used, vulnerable check permission function, broadcast without receiver check permission, WebView content and file access, and 'Phone number or IMEI detected'.

Similarly, 'JavaScript enabled in WebView' threat which means JavaScript is enabled for this application in the WebView, so the possibility of XSS (cross-site scripting) attacks increases if JavaScript is not disabled.

5. Discussion and Analysis

The security analysis of the three applications, using the security testing tools MobSF and QARK, has identified possible vulnerabilities. These vulnerabilities are categorized and presented in Table.7.

Issue	App 'A'	App 'B'	App 'C'
Hash Algorithm	SHA1	SHA256	SHA256
Certificate algorithm vulnerable to hash collision	✓		
Application domains	✓		
Enabled Clear-text traffic for App	✓	✓	
Insecure WebView Implementation, accepting any SSL Certificate and ignoring errors	✓		✓
Enabled Remote WebView debugging			✓
Privacy Trackers			✓

Table.7: High-risk findings for the applications A, B, and C.

Upon comparing the results across the three applications, it was found that App 'A' is the least secure due to its use of the SHA1 hashing algorithm. Both apps 'A' and 'B' have clear text network traffic enabled between the application and the server, which exposes their data to potential network attackers. Apps 'A' and 'C' have insecure WebView implementation issues, indicating a high-risk vulnerability due to their handling of SSL certificates and the communication security.

App 'C' allows remote WebView debugging, which could potentially enable attackers to execute arbitrary code.

The vulnerabilities identified in the three applications can lead to the exposure of their API keys. The API keys for app 'A' and app 'C' were easily accessible as they were directly stored in the manifest file. App 'B', on the other hand, had additional restrictions

on its platform console, making it more challenging to obtain the API key. However, with reverse engineering, it could be extracted.

As for the permissions of the applications, which were identified as having a dangerous status. Table.8 provides a summary of the similarities and differences between the three apps regarding these permissions. Apps 'A' and 'C' grant permission to pinpoint the user's exact location. This is a highly dangerous permission and is not necessary for the applications' functionality, so it should be removed from granted permissions for both apps. Another dangerous permission granted by the three apps is the ability to record audio. This permission allows the application to access the device's microphone without the user's knowledge at any time. Also, all three apps grant access to local storage to read and write data on the user's device. This is considered a dangerous permission because it allows the app to modify, copy, or even delete critical data for the user.

Apps 'A' and 'C' grant the permission to read contact information on a customer's device. All three apps grant the permission read phone state. App 'B' requests the system alert window permission, which enables the app to create windows that are displayed on top of all other apps.

Finally, app 'C' grants the permission to write contacts, which enables the app to access and modify contact information on the customer's device.

PERMISSION	App 'A'	App 'B'	App 'C'
'ACCESS_COARSE_LOCATION'	✓	✓	✓
'ACCESS_FINE_LOCATION'	✓		✓
'RECORD_AUDIO'	✓	✓	✓
'READ_EXTERNAL_STORAGE'	✓	✓	✓
'WRITE_EXTERNAL_STORAGE'	✓	✓	✓
'READ_CONTACTS'	✓		✓
'CAMERA'	✓	✓	✓
'READ_PHONE_STATE'	✓	✓	✓
'SYSTEM_ALERT_WINDOW'		✓	
'WRITE_CONTACTS'			✓

Table.8: The Apps 'A', 'B, and 'C' Dangerous Permissions

6. Conclusion and Future Work

We were able to identify and categorize potential vulnerabilities, including issues with hashing algorithms, clear text network traffic, insecure WebView implementation, remote WebView debugging, and privacy trackers. We found that the API keys for two of the applications were easily accessible, and even the more secure application's API key could be obtained through reverse engineering. This highlights the need for more secure methods of storing and accessing these keys.

Furthermore, our analysis of the applications' permissions revealed several dangerous permissions granted by the apps. These include access to the user's exact location, the ability to record audio, access to local storage, and access to contact information. These permissions, if misused, could lead to serious privacy violations and data breaches.

In conclusion, this research underscores the Importance of robust security measures in application development. It is crucial for developers to know the potential vulnerabilities and take necessary steps to mitigate them. This includes using secure hashing algorithms, disabling clear text network traffic, ensuring secure WebView implementation, disabling remote WebView debugging, and carefully managing privacy trackers. Additionally, permissions should be granted reasonably and only when necessary for the app's functionality. Users should be informed about the data being collected, how it is used, and with whom it is shared.

Acknowledgment: We would like to thank PTUK University for providing this research article with essential financial support. Your support for academic research has been instrumental in making this paper a success.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers.

References

[1] Android Developers. (n.d.-a). Retrieved April 2, 2025, from https://developer.android.com/privacy-and-security/security-best-practices

[2] Android Developers. (n.d.-b). Retrieved April 2, 2025, from https://developer.android.com/studio

[3] Dar, M. A. (2017). A novel approach to enhance the security of android based smart phones. Proceedings of 2017 International Conference on Innovations in Information, Embedded and Communication Systems, ICIIECS 2017, 2018-January, 1–5. https://doi.org/10.1109/ICIIECS.2017.8275923

[4] Gong, A., Zhong, Y., Zou, W., Shi, Y., & Fang, C. (2020). Incorporating Android Code Smells into Java Static Code Metrics for Security Risk Prediction of Android Applications. Proceedings - 2020 IEEE 20th International Conference on Software Quality, Reliability, and Security, QRS 2020, 30–40. https://doi.org/10.1109/QRS51102.2020.00017

[5] Khadiranaikar, B., Zavarsky, P., & Malik, Y. (2017). Improving Android application security for intent based attacks. 2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference, IEMCON 2017, 62–67. https://doi.org/10.1109/IEMCON.2017.8117149

[6] Li, D., Wang, Z., & Xue, Y. (2018). Fine-grained Android Malware Detection based on Deep Learning. 2018 IEEE Conference on Communications and Network Security (CNS), 1–2. https://doi.org/10.1109/CNS.2018.8433204

[7] Li, R., Diao, W., Li, Z., Yang, S., Li, S., & Guo, S. (2022). Android Custom Permissions Demystified: A Comprehensive Security Evaluation. IEEE Transactions on Software Engineering, 48(11), 4465–4484. https://doi.org/10.1109/TSE.2021.3119980

[8] Meng, X., Qian, K., Lo, D., & Bhattachrya, P. (2018). Detectors for Intent ICC Security Vulnerability with Android IDE. International Conference on Ubiquitous and Future Networks, ICUFN, 2018-July, 355–357. https://doi.org/10.1109/ICUFN.2018.8436802

[9] MobSF. (n.d.). Retrieved April 2, 2025, from https://github.com/MobSF/Mobile-Security-Framework-MobSF

[10] MobSF Manual. (n.d.). Retrieved April 2, 2025, from https://hacken.io/discover/static-analysis-of-android-mobile-applications-mobsf-manual/

[11] Nirumand, A., Rajaei, Z., Zamani, B., & Kolahdouz-Rahimi, S. (2020). Modeling Android Security using an Extension of Knowledge Discovery Metamodel. 2020 10h International Conference on Computer and Knowledge Engineering, ICCKE 2020, 356–361. https://doi.org/10.1109/ICCKE50421.2020.9303707

[12] qark. (n.d.). Retrieved April 2, 2025, from https://github.com/linkedin/qark

[13] Rahman, A., Pradhan, P., Partho, A., & Williams, L. (2017). Predicting Android Application Security and Privacy Risk with Static Code Metrics. Proceedings - 2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems, MOBILESoft 2017, 149–153. https://doi.org/10.1109/MOBILESoft.2017.14

[14] Sarkar, A., Goyal, A., Hicks, D., Sarkar, D., & Hazra, S. (2019). Android Application Development: A Brief Overview of Android Platforms and Evolution of Security Systems. Proceedings of the 3rd International Conference on I-SMAC IoT in Social, Mobile, Analytics and Cloud, I-SMAC 2019, 73–79. https://doi.org/10.1109/I-SMAC47947.2019.9032440

[15] Wu, S., Zhang, Y., Jin, B., & Cao, W. (2017). Practical static analysis of detecting intent-based permission leakage in Android application. International Conference on Communication Technology Proceedings, ICCT, 2017-Octob, 1953–1957. https://doi.org/10.1109/ICCT.2017.8359970

[16] Yadav, S., Apurva, A., Ranakoti, P., Tomer, S., & Roy, N. R. (2017). Android vulnerabilities and security. 2017 International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN), 204–208. https://doi.org/10.1109/IC3TSN.2017.8284477