

Security Analysis of Android Applications for Hotel and Flight Booking Applications

Vatcharavaree Wongsuna, Assoc. Prof. Sudsanguan Ngamsuriyaroj

Faculty of Information and Communication Technology, Mahidol University, Thailand

vatcharavaree.wog@student.mahidol.ac.th, sudsanguan.nga@mahidol.ac.th

Abstract— The tourism industry's exponential growth has currently led to an increase in hotel and flight bookings, especially through mobile applications. There are various tools and platforms to build a mobile application, and each mobile function may need specific permissions to access certain features of a mobile device, and that would make devices and applications vulnerable. In this work, we aim to conduct a security analysis of many Android applications for hotel and airline bookings. Using an open-source tool, the security-related features of 20 Android apps are extracted from the domains and divided into six groups, including security mechanisms, shared Android components, dangerous privilege calls, and three code-related groups. Those applications were then clustered according to their features, and we found that hotel applications had Android security features, such as the RSA algorithm and detecting jailbreak. Furthermore, third-party apps were more likely to share Android components with other apps on the same device. Future research could focus on analyzing Android apps for hotel or airline booking exclusively and expanding the dataset size to identify and analyze patterns effectively.

Keywords— Android Application Security, security Analysis

I. INTRODUCTION

Mobile phones and other smart devices, such as tablets, have evolved from being mere communication devices to becoming more functional. They now offer features such as camera capabilities, mapping and location services, network connectivity, and the ability to record audio. Moreover, these features are used by mobile applications to serve the business needs. So, they may require accessing and compromising some sensitive features such as camera, microphone, or storage. This could have far-reaching implications in terms of privacy and security.

According to CVEdetails [1], Android was found to be the most vulnerable product in 2017. The vulnerabilities reported were mainly related to overflow, code execution, and information disclosure which can compromise sensitive information. Moreover, Pradeo's report [2] revealed that over 60% of mobile applications failed to prevent data leakage or corruption, even though they had policies and permissions in place. The study also identified 75% of vulnerabilities listed in the OWASP top ten flaws and requested more permission than the app's purpose.

Therefore, this research aims to conduct a security analysis using an automatic tool to extract security-related features from 20 well-known Android applications for hotel and flight

booking. We do the static analysis technique using open-source tools and eventually perform the clustering analysis using a data mining tool such as RapidMiner [13] to identify any security patterns. However, there are some scopes of this work as outlined below.

- This research will perform only static analysis techniques to assess the application information. The analysis will involve examining the application's source code, manifest.xml file, and the security features implemented by the apps.
- Clustering analysis will be performed using the Rapid Miner data mining tool with the K-means algorithm and elbow method.
- The OWASP MSTG [4] will be used as a reference in this research, but not all aspects will be covered.
- The dataset consists of 20 Android applications from the travel category. These applications can be categorized into three types: applications that provide booking services for both hotel rooms and flights, applications developed by airline companies, and applications developed by hotels.
- Last, these 20 Android applications are not the latest versions and mostly date back to the years 2010 to 2021.

The remainder of this paper is organized as follows: Section 2 presents the related work. Section 3 describes the security analysis. Section 4 explains the Android applications. Section 5 depicts our proposed work. Section 6 presents the analysis results, and the concluding remarks are given in Section 7.

II. RELATED WORK

There are various approaches to analysing an Android application. Each approach serves a specific purpose and provides insights into different aspects of the app. Some common approaches to analysis include security analysis static analysis, dynamic analysis, and penetration testing. As in the paper [5], they analyse and identify vulnerabilities, and malicious Android apps using vulnerability, static, and dynamic analysis approaches for both Android apps and Android platforms. They introduce an Analysis Framework for Security Analysis of Android applications and Android platforms to provide effective way and sustainable security solutions.

Furthermore, they implemented OWASP Droid Fusion [14], OWASP Mobile Top Ten, National Vulnerabilities Database (NVD) [15], and Open-Source Vulnerabilities Database

(OSVDB) [16] as the guidelines. They also decompiled the target Android app, analysed the source code, and extracted features to compare with the malware signature. In addition, they monitored and analysed the application's behaviours, data traffic, and application logs. If the application was not malicious, they will discover vulnerabilities and apply the available patches before recompiling and signing the app to perform dynamic analysis.

The paper in [6] did a privacy analysis on an Android app, as well as combined the control of the OWASP Mobile Security Project [4], Open Android Security Assessment Methodology (OASAM) [17], and other good practices to introduce a methodology that involves both static and dynamic analysis. They collect information on Android operating systems before and after installation of an application, to determine possible data leakage. They also examine the login process to identify potential vulnerabilities and intercept transactions between the app and server to verify data encryption. Once everything is completed, they perform application source code analysis and component analysis to identify hidden vulnerabilities. The results of the study reveal that the critical vulnerabilities are related to application development, which is the responsibility of developers to be aware of and protect user data.

Another research that does the privacy and security analysis is paper [7]. This work's purpose is to compare the security and privacy outlines based on the differences of the applications. They utilized the OWASP Mobile Top Ten to examine common vulnerabilities and conducted both static and network analysis. They classified threats identified by a tool named DroidSafe into categories, then mapped with OWASP. However, their work found that the Static Analysis failed to examine the functional implication of the threat's prone component, so they decided to perform a manual analysis instead.

III. SECURITY ANALYSIS

Security analysis is a method of evaluating and assessing the security of an application to identify any vulnerabilities, weak points, or potential threats. The goal is to recommend measures for mitigating or reducing these risks. It usually uses automated tools to examine and analyse the application, and then generate a report. There are two common analysis process that can be performed: static and dynamic analysis.

Static analysis involves reviewing the application's source code to find potential security vulnerabilities and common issues, such as insecure data storage or improper input validation. This is typically carried out by an automated security scanner like MobSF [11], which decompiles the APK file to analyse the app's bytecode and resources. Checking the Manifest file and then a source code review is conducted to identify code-level vulnerabilities and generate a report.

On the contrary, dynamic analysis is performed while running an Android app. The main objective is to find any weak points in an application and verify security mechanisms that provide sufficient protection against the most prevalent

type of attacks, such as authentication and authorization issues, exposing data in transit, etc. Moreover, testing of this analysis may vary depending on the mobile application type. Table 1 shows actions that may be involved in both analyses.

TABLE 1. SAMPLE ACTIONS FOR STATIC AND DYNAMIC ANALYSIS

Static analysis	Dynamic Analysis
• Information Gathering / General Information, • Decompile to java and smali (Reverse Engineering), • Permission Analysis, • Manifest Analysis, and • Application's source code review	• Capture HTTP/HTTPS Traffic, Transport Layer Testing, • Memory Analysis, • File Analysis on application data, • Server-Side Attacks • Activity Tester, • Logcat and Dumpsys • Input/output validation (cross-site scripting, SQL injection, etc.)

IV. ANDROID APPLICATION

An Android application is application software running on the Android operating system, which is a Linux-based open-source platform developed by Google. To run an app on the Android device, Android SDK tools will compile application source code with its resources into a single file format named Android Package Kit (APK), which is an archive or zip file.

A. Android Application Components

There are four main components to build an Android application: Activities, Services, Broadcast Receivers, and Content providers.

1) Activity: A screen with the user interface (UI) for a user to interact with. In addition, when creating an app, the developer must declare every activity inside the Android Manifest file. Otherwise, the activity is not authorized and allowed to be carried out by the user.

2) Service: A component that allows an application to perform long-running operations in the background, such as notifications. These services must be declared in the AndroidManifest.xml file. There are two types of services: Started services and Bound services. Started services are initiated by calling `startService()` and can run indefinitely until they are stopped. On the other hand, Bound services are initiated by binding to them via `bindService()`. They are tightly coupled to the application and can be used to provide a client-server interface between the application and the service.

3) Broadcast Receiver: A component that allows an application to receive notifications from other apps and the system. The broadcast receiver must be declared in the AndroidManifest.xml file with an associated `<intent-filter>` to specify the action or event that the receiver would like to react to or receive. Without this declaration, the app will not be able to listen or react to the broadcast messages. It's an important component for ensuring smooth communication between different parts of an Android system.

4) Content Provider: It manages a shared set of application data that is stored in various locations such as the file system, SQLite database, or other persistent storage location. This enables other applications to access and modify the data, provided they have been granted permission by the content provider.

B. Permissions and Protection Level

Android provides predefined permission for Android developers to protect user's privacy. If an app needs to access sensitive data or potentially dangerous system features like location or storage, the application must first request permission from the user. The user must then approve the request before the app can proceed. This is an important measure to ensure that users have control over their own data and can make informed decisions about how it is accessed and used.

In addition, Android permissions are categorized into different levels of protection, including normal, dangerous, and signature protection levels.

1) Normal Permission level: This refers to the basic services that Android applications use, including Bluetooth, Internet, and Vibration. When an Android app declares its permissions in a manifest file, they are typically classified as PROTECTION_NORMAL. At install time, the operating system automatically grants these permissions, and the user cannot revoke them.

2) Dangerous Permission level and Permission group: This refers to features that are protected by Android. In case an app requests access to one or all API calls under a permission group, the system will automatically grant access to all APIs under that group. For example, an app declares access to the "coarse location" in the AndroidManifest file, it can access all API calls under the location permission group. In addition, if the app calls any API under the Location permission group again in the future, the system will automatically grant access without asking for permission from the user. So, this is essential to be mindful of the permissions an app requests and understand the risks associated with granting access to potentially sensitive data.

3) Signature Permission: There are certain permissions that are granted at the installation time. These permissions are signed with the same certificate as the app itself, to ensure that they are legitimate and safe to use.

C. Google Primary Security Services for Android

Google also provides some security services to protect users and legitimate apps from malicious applications such as Verify Apps, SafetyNet, SafetyNet Attestation, and Android Device Manager. These services are not part of the Android Open-Source Project, but they are integrated into several Android devices shown in Table 2.

TABLE 2. GOOGLE PRIMARY SECURITY SERVICES

Google Play Security Services	Description
Verify Apps	It warns users or automatically blocks attempted installation of any harmful apps. This also continually scans applications on device.
SafetyNet	It provides a set of services and APIs that help protect an application against security threats, including device tampering, bad URLs, potentially harmful apps, and fake users.
SafetyNet Attestation	Third-party API to determine whether the device is CTS compatible. Attestation can also assist identify the Android app communicating with app server.
Android Device Manager	A web app and Android app to locate lost or stolen devices.

V. PROPOSED WORK

Figure 1 demonstrates an overview of the process of our proposed work. We selected Android apps widely used in Thailand for hotel and airline booking and conducted a security analysis by using an automated scanner named MobSF [11] to analyze and extract security-related data from the target application. In addition, we manually mapped results with MSTG [4] for some parts. Then we categorized the features into six groups and clustered to see some patterns. Finally, we manually calculated the centroid table and found the best K-value using the Elbow method. Table 3 shows some steps we performed during the static analysis.

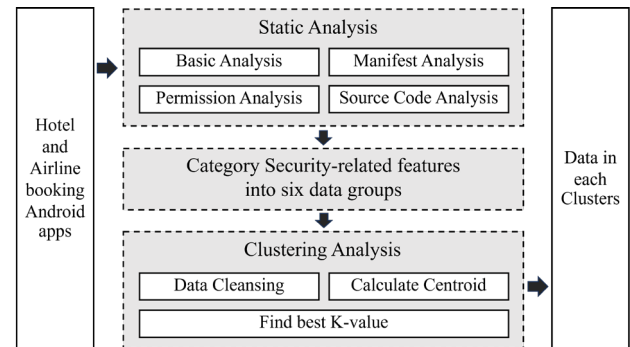


Figure 1. System overview diagram

A. Static Analysis

Four subprocesses in the static analysis are Basic analysis, Manifest analysis, Permission analysis and Source Code analysis. The results are obtained from MobSF analysis report.

TABLE 3. DETAILS OF TOOLS AND STEPS TO PERFORM STATIC ANALYSIS

Basic Analysis		
Tool	How to perform	Output Information
MobSF	collect information from MobSF analysis report under Information section.	General Information - Application Name - Application Version - Target SDK version - Min SDK version support - Category - Package Name

APKTool v.2.4.1	Decompile an APK file, then search for platform folder	Security Features - Root Detection Capability
MobSF	Decompile an APK file, then search for platform folder	Security Features - Signature Verification Algorithm - Signature Algorithm Status
MobSF	Check root detection capability by searching word "Root Detection" inside report.	- SafetyNet API Implemented
Manifest Analysis		
Tool	How to perform	Output Information
MobSF	Collect information on MobSF analysis report under info. and Manifest analysis section.	Android Components that are not protected. - Exported Activity - Export Services - Exported Broadcast Receiver - Exported Content Provider
MobSF	check if the application allows the attacker to perform data Backup or not.	- Check if "android:allowBackup=true"
Permission Analysis		
Tool	How to perform	Output Information
MobSF	Collect info under permission analysis section	- List of Normal Permissions - List of Protected Permission
Source Code Analysis		
Tool	How to perform	Output Information
MobSF	We collect the information from the MobSF analysis report under the Code Analysis section.	- Type of code issues - Severity - Threats
Manual	Search the vulnerable code in the java files.	- Vulnerable code
Manual	Map result with OWASP Mobile Top Ten and the MSTG Checklist	- Summary of code issues map to MSTG and OWASP top 10 2016. - CVSS scores

B. Defined Parameters

This section will talk about six data groups that we classified.

1) Data Group 1: it's about Android security features used by an application. Regarding the MSTG Android Anti-Reversing Defenses [8], we will focus on root detection, SafetyNet implemented, and file integrity checks.

2) Data Group 2: This group focuses on "exported" Android components (activity, service, or content) that can be accessed by other apps.

3) Data Group 3: This group focuses on app permissions that are considered dangerous. To evaluate whether the permissions requested are related to the app's purpose or not.

4) Data Group 4: This group focus on analysing Android Manifest file and check for the ADB full back up support, copies data to the clipboard or not, and check whether app requests root privileges.

5) Data Group 5: it's about others source code issues with the CVSSv2 (Common Vulnerability Scoring System).

6) Data Group 6: This group is an others source code issues related to WebView and SQL. This group focuses on other source code issues that did not have CVSS scores.

C. Clustering of output parameters

In this work, we use the K-mean algorithm with the centroid models. We select Euclidean Distance to calculate the distance between two points and find the best K value using Elbow Curve [$\sqrt{\sum (A_i - B_i)^2}$] methods. After getting the results of the centroid table from the RapidMiner tool, we calculated using k-value from 2 to 6 clusters. Then we use this value to calculate the Within-Cluster Sum of Square or WCSS, which is the sum of the squared distance between each point and the centroid in the cluster.

VI. ANALYSIS RESULTS

This section will discuss the results of our analysis. Table 4 shows the result we extracted during static analysis phase before clustering. After doing the data preparation, import to the RapidMiner tool, and calculate the WCSS scores using Elbow curve method. We found that the best k-value is 4 clusters.

A. Output Analysis for Each Data Group

To see some pattern or relationship in which the application may be similar or completely different, we are assigning a colour to each cluster and then summarize them based on the parameters or attributes, which can group just some of them.

1) Data Group 1 (Android Security Features): Third-party apps who's capable to book both hotels and airlines. Likely to implement similar Android security features as shown in cluster 0, while applications developed by airlines or hotels are classified in cluster 1. And once we are looking more closely at the parameters or attributes, seems like those applications in Cluster 1 are implementing Android security features more than apps in Cluster 3, either. in terms of algorithms for signing applications and detecting jailbreak devices. Moreover, the application in cluster 3 implements a low number of Android security features also the application was signed with "SHA1 with RSA" whose SHA1 hash algorithm is known to have collision issues.

2) Data Group 2 (Attack Surface Exported Android Components): In this group, the distribution of applications from different types which may result from platforms or tools for development. Causing the results to come out so different that no conclusions can be drawn. In addition, the results show that applications in cluster 2 are more likely to share Android components (Activities, Services, Content Providers, and Broadcast Receivers) with other apps in the same device.

3) Data Group 3 (Dangerous Permission Requested): In this group, there is no application that can be identified to a particular cluster. In addition, from the results, the app parameter shows that most applications request access to storage and location. We also see that apps in cluster 1 request more dangerous permissions than applications in other clusters,

TABLE 4. STATIC ANALYSIS RESULTS FOR 6 DATA GROUPS

#	Data Group	Application Name	OWASP MASVS	OWASP Top 10	Third Party							Airline						Hotel						
					T1	T2	T3	T4	T5	T6	T7	A1	A2	A3	A4	A5	A6	H1	H2	H3	H4	H5	H6	H7
1	Android Security Features	Root Detection	-	-	Y	Y	Y	N	Y	Y	N	Y	Y	N	N	Y	N	Y	N	Y	N	N	N	Y
		SafetyNet	-	-	Y	N	Y	N	N	N	N	N	N	N	N	N	N	Y	N	N	N	N	N	N
		Signature Algorithm	-	-	SHA1	SHA1	SHA1	MD5	SHA1	SHA1	SHA1	SHA256	SHA256	SHA1	SHA1	SHA256	SHA1	SHA256	SHA256	MD5	SHA256	SHA256	SHA256	SHA256
		Signer Certificate Status	-	-	Bad	Warning	Warning	Good	Warning	Warning	Bad	Good	Good	Warning	Bad	Good	Warning	Good	Good	Good	Good	Good	Warning	Warning
2	Attack Surface Exported Android Components	Activity	-	-	1.4	7.19	6.45	16.67	1.11	1.55	3.7	13.64	0	6.25	0	0	0	6.67	0	0	0.84	42.86	15.38	2.46
		Service	-	-	33.33	18.6	36.36	55.56	12.5	26.32	16.67	35.71	14.29	0	0	14.29	0	14.29	0	9.09	11.11	0	36	16.67
		Broadcast Receiver	-	-	53.85	50	68.75	66.67	68.75	53.85	40	57.14	50	50	100	60	100	66.67	0	55.6	33.33	33.33	54.55	21.43
		Content Provider	-	-	0	0	0	0	0	0	0	25	0	0	100	0	0	0	0	0	0	0	0	0
3	Dangerous Permission Requested	Calendar	-	-	Y	N	N	N	N	Y	N	Y	N	Y	N	N	N	N	N	N	N	N	N	Y
		Call Log	-	-	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
		Camera	-	-	Y	Y	N	N	N	Y	N	Y	Y	Y	N	Y	N	N	N	N	N	N	Y	Y
		Contacts	-	-	N	N	Y	N	N	Y	N	Y	N	Y	N	N	N	Y	N	N	Y	N	N	N
		Location	-	-	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y
		Microphone	-	-	N	N	N	N	Y	N	N	Y	N	Y	N	N	N	Y	N	N	N	N	N	N
		Phone	-	-	N	N	N	N	Y	Y	N	Y	N	Y	N	Y	N	Y	N	Y	Y	N	N	N
		Sensors	-	-	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
		SMS	-	-	N	N	N	N	N	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N
		Storage	-	-	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y
4	Source Code Issues related to data backup and Root privilege	Backup app Data	-	-	Y	N	N	N	N	N	Y	N	Y	Y	Y	N	Y	Y	Y	Y	N	Y	N	N
		copies data to clipboard	MSTG-Storage	-	Y	N	N	N	Y	Y	N	N	N	N	N	N	N	N	N	N	Y	Y	Y	N
		request ROOT privileges	MSTG-Resilience-1	-	N	N	N	N	Y	Y	N	Y	N	N	N	N	N	N	N	N	N	N	N	N
5	Others Source Code Issues with the CVSSv2	Insecure Implementation of SSL	MSTG-Network-3	M3	7.4	7.4	0	0	7.4	0	0	7.4	0	0	7.4	7.4	0	0	0	0	0	0	0	7.4
		WebView ignores SSL cert error & accept any SSL cert	MSTG-Crypto-6	M5	7.5	0	0	0	0	7.5	7.5	7.5	0	0	0	0	0	0	0	0	0	0	0	0
		read/write external storage	MSTG-Storage-2	M2	5.5	5.5	5.5	5.5	5.5	5.5	5.5	5.5	5.5	5.5	0	0	5.5	5.5	0	5.5	5.5	5.5	5.5	5.5
		The App log information	MSTG-Storage-3	-	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5
		create TEMP file	MSTG-Storage-2	M2	5.5	5.5	0	5.5	5.5	5.5	0	5.5	5.5	0	5.5	0	5.5	5.5	0	5.5	0	5.5	0	5.5
		write to App Directory	MSTG-Storage-14	-	0	0	3.9	0	0	0	0	0	0	0	0	0	0	3.9	0	0	0	0	0	3.9
		File is World Readable	MSTG-Storage-2	M2	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0
		IP Address disclosure	MSTG-Code-2	-	4.3	4.3	4.3	4.3	4.3	4.3	4.3	4.3	4.3	0	4.3	4.3	4.3	4.3	0	0	4.3	4.3	4.3	4.3
		Hardcode sensitive information	MSTG-Storage-14	M9	7.4	7.4	7.4	7.4	7.4	7.4	7.4	7.4	7.4	7.4	7.4	7.4	7.4	7.4	0	7.4	7.4	7.4	7.4	7.4
		uses ECB mode	MSTG-Crypto-2	M5	0	0	0	0	5.9	0	0	5.9	0	0	0	0	0	0	0	0	0	0	5.9	5.9
		Insecure hash function - MD5	MSTG-Crypto-4	M5	7.4	7.4	7.4	7.4	7.4	7.4	7.4	7.4	0	0	0	7.4	7.4	7.4	0	7.4	7.4	7.4	7.4	7.4
		Insecure hash function - SHA-1	MSTG-Crypto-4	M5	5.9	5.9	5.9	5.9	5.9	5.9	5.9	5.9	5.9	0	0	0	5.9	5.9	0	5.9	5.9	5.9	5.9	5.9
		JAVA Hash Code	MSTG-Crypto-4	-	2.3	2.3	2.3	2.3	2.3	2.3	2.3	2.3	2.3	2.3	2.3	2.3	2.3	2.3	0	2.3	0	0	0	0
6	Others Source Code Issues related to WebView and SQL	Insecure Random Generator	MSTG-Crypto-6	M5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	0	7.5	7.5	7.5	7.5	7.5
		Execution of user-controlled code in WebView	MSTG-Platform-7	M1	8.8	8.8	0	8.8	0	8.8	0	8.8	8.8	0	0	8.8	8.8	8.8	0	8.8	8.8	0	8.8	8.8
		uses SQLite & execute raw SQL query	-	M7	5.9	5.9	5.9	5.9	5.9	5.9	0	5.9	5.9	5.9	5.9	5.9	5.9	5.9	0	5.9	5.9	5.9	5.9	5.9
		Remote WebView debugging is enabled	MSTG-Resilience-2	M1	0	0	5.4	0	0	0	5.4	5.4	0	0	0	0	0	5.4	0	0	0	0	5.4	5.4

requesting access to contact, camera, microphone, phone, or even SMS.

4) Data Group 4 (Source Code Issues related to data backup and Root privilege): In this group, the results shows that those apps in cluster 3 do not have the ability to backup app data / copy data and request higher permissions. It is considered good in terms of security. In addition, applications in Cluster1 can backup app data, copy data to the clipboard, and request root privileges. Mean these apps may have higher risks than others.

5) Data Group 5 (Others Source Code Issues with the CVSSv2): In this group, we can conclude into 3 groups. The first group is Cluster 0 and Cluster 1 which are different from SSL and ECB modes. Second is cluster 2, which is different from all other groups. The third group is Cluster 3 who's like Cluster 0 except hash algorithm, SHA-1 / md5, and Java hash.

6) Data Group 6 (Others Source Code Issues related to WebView and SQL): In this group, the applications in Clusters 0 and 1 are different in the "remote web view" attribute. From the security point of view, the apps in such clusters are more vulnerable than the apps in other clusters.

VII. CONCLUSIONS

In this paper, we perform the security analysis of 20 android applications for hotel and airline reservation. We divided the security features of such applications into 6 groups, and we do the clustering on them. Eventually, we found the results listed below.

- For data group 1, applications in cluster 1 implemented android security features than apps and have a good status of signer certificate except H7. While apps in cluster 3 implement a low number of Android security features also the app was signed with "SHA1 with RSA".
- For data group 2, applications in cluster 2 are more likely to share Android components with other apps in the same device. Moreover, apps in cluster 1, are more likely to allow other applications to query or modify the data.
- For data group 3, apps in cluster 1 request more dangerous permissions than applications in other clusters, requesting access to contact, camera, microphone, phone, or even SMS. While applications in clusters 0 request permission only they needed.
- For data group 4, only applications in clusters 1 requested root privileges. While applications in cluster 2 and some of applications in cluster 0 allow to backup applications data.
- For data group 5, applications in cluster 2 have a low number of source code issues such as apps log information, create TEMP file, etc.
- Last, data group 6, applications in cluster 1 have the code issues related to remote WebView, execute raw SQL query and execution of user-controlled code in

WebView. While applications in other clusters have some of them.

In addition to the data analysis presented in this work, we encountered some limitations. Firstly, the number of applications used was small and resulted in a limited range of clustering such that only a few patterns could be identified. Secondly, the versions of the applications used were not the latest version; thus, some security features may have been updated, and the analysis may not fully reflect the current status. Thirdly, the number of clusters was determined based on the Elbow Curve graph, and we chose four clusters for all data groups, but this may not be suitable for all groups. To achieve better meaningful results, future research may consider analysing the same type of Android applications or the same group, such as apps exclusively for hotel bookings or airlines, so that we can facilitate detailed comparisons. In addition, expanding the dataset size and using the latest version of the applications would enhance the ability to identify and analyse patterns effectively. Finally, each data group should have a different number of clusters in the analysis.

ACKNOWLEDGMENT

This research work was partially supported by the Faculty of Information and Communications Technology, Mahidol University.

REFERENCES

- [1] "Top 50 Products by Total Number of Distinct Vulnerabilities in 2017". [Online]. Available: <https://www.cvedetails.com/>.
- [2] Pradeo, Mobile Applications Threats Review for S1 2017 pages 4-10. 2017.
- [3] Google, Android Security 2017 Year in Review Final Report pages 15, 2017.
- [4] OWASP Mobile Application Security. [Online]. Available: <https://owasp.org/www-project-mobile-app-security/>.
- [5] Umasankar, Analysis of Latest Vulnerabilities in Android, 2017.
- [6] Alejandro Argudo, Gabriel López, Franklin Sánchez, Privacy Vulnerability Analysis for Android Applications, 2017.
- [7] Ashish Rajendra Sai, Jim Buckley, Andrew Le Gear, Privacy and Security analysis of cryptocurrency mobile applications, 2019.
- [8] "Permissions on Android". [Online]. Available: <https://developer.android.com/guide/topics/permissions/>
- [9] Sen Chen, Yuxin Zhang, Lingling Fan, Jiaming Li, Yang Liu, AUSERA: Automated Security Vulnerability Detection for Android Apps, 2022.
- [10] Pasquale Stirparo, Ioannis Kounelis, The mobileleak project: Forensics methodology for mobile application privacy assessment, 2013
- [11] "Mobile Security Framework (MobSF)". [Online]. Available: <https://github.com/MobSF/Mobile-Security-Framework-MobSF>.
- [12] "Android Testing Cheat Sheet". [Online]. Available: https://owasp.org/index.php/Android_Testing_Cheat_Sheet
- [13] RapidMiner. [Online]. Available: <https://rapidminer.com/>
- [14] OWASP Droid Fusion. [Online]. Available: https://wiki.owasp.org/index.php/OWASP_Droid_Fusion
- [15] OWASP Mobile Top Ten. [Online]. Available: <https://owasp.org/www-project-mobile-top-10/>
- [16] Open-Source Vulnerabilities Database (OSVDB). [Online]. Available: <https://osv.dev/>
- [17] Open Android Security Assessment Methodology (OASAM). [Online]. Available: <https://github.com/b661/OASAM>