



Article

A Risk Assessment Framework for Mobile Apps in Mobile Cloud Computing Environments

Noah Oghenefego Ogwara ¹, Krassie Petrova ^{1,*} , Mee Loong Yang ¹ and Stephen G. MacDonell ²

¹ Department of Computer Science and Software Engineering, School of Engineering, Computer and Mathematical Sciences, Auckland University of Technology, City Campus, Auckland 1010, New Zealand; fego.ogwara@macvad.com (N.O.O.); bobby.yang@aut.ac.nz (M.L.Y.)

² Centre for Data Science and AI, School of Engineering and Computer Science, Faculty of Engineering, Victoria University of Technology, Kelburn Campus, Wellington 6012, New Zealand; stephen.macdonell@vuw.ac.nz

* Correspondence: krassie.petrova@aut.ac.nz

Abstract: Mobile devices (MDs) are used by mobile cloud computing (MCC) customers and by other users because of their portability, robust connectivity, and ability to house and operate third-party applications (apps). However, the apps installed on an MD may pose data security risks to the MD owner and to other MCC users, especially when the requested permissions include access to sensitive data (e.g., user's location and contacts). Calculating the risk score of an app or quantifying its potential harmfulness based on user input or on data gathered while the app is actually running may not provide reliable and sufficiently accurate results to avoid harmful consequences. This study develops and evaluates a risk assessment framework for Android-based MDs that does not depend on user input or on actual app behavior. Rather, an app risk evaluator assigns a risk category to each resident app based on the app's classification (benign or malicious) and the app's risk score. The app classifier (a trained machine learning model) evaluates the permissions and intents requested by the app. The app risk score is calculated by applying a probabilistic function based on the app's use of a set of selected dangerous permissions. The results from testing of the framework on an MD with real-life resident apps indicated that the proposed security solution was effective and feasible.

Keywords: mobile cloud computing; security threats; risk assessment; mobile device; mobile application; application security; data protection; machine learning; ensemble model



Citation: Ogwara, N.O.; Petrova, K.; Yang, M.L.; MacDonell, S.G. A Risk Assessment Framework for Mobile Apps in Mobile Cloud Computing Environments. *Future Internet* **2024**, *16*, 271. <https://doi.org/10.3390/fi16080271>

Academic Editor: Yuezhi Zhou

Received: 2 July 2024

Revised: 24 July 2024

Accepted: 26 July 2024

Published: 29 July 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Mobile devices (MD) are popular because of their portability, robust connectivity, and capability to house and operate different third-party applications. In their normal course of activities, MD users may connect to several cloud services and thus become part of the mobile cloud computing (MCC) environment. MCC refers to the concept of providing users with flexible, reliable, and anywhere-and-anytime access to cloud-stored data and other cloud services. It combines the powerful resources of cloud computing (CC) and wireless network technologies [1]. However, some of the connected networks forming the mobile cloud may not be secure and may expose the applications (apps) that reside on the MD to security threats (e.g., breaches of data integrity, confidentiality, and service availability [2]). In addition, when personal MDs are used in the workplace, there is a possibility of introducing security threats to corporate networks [3].

Furthermore, many of the apps installed on MDs are inherently risky due to the broad range of permissions they request to operate, such as access to user location data, user contacts, or photo galleries. Apps downloaded from reputable app markets such as Apple's App Store and Google Play may not be intentionally malicious, but if excessive permissions are granted, personal information, sensitive data, and user privacy can nevertheless be compromised [4]. For example, when an end-user MD is part of an Internet of Things (IoT) ecosystem, a malicious app that resides on one device may negatively affect other connected

devices. The threats to the IoT environment related to mobile app vulnerabilities have been mostly associated with devices using the Android mobile operating system [5] due in part to the open publication policy that allows users to download apps from both official and unofficial market stores. In turn, malware developers have shifted their attention to targeting apps that can be deployed on such open platforms [6].

Android app developers do not always consider the potentially harmful effects of requesting multiple permissions for effective app operation, and more specifically, how the requested permissions can be manipulated and misused to breach user privacy [7]. In more recent Android versions, the permission system affords users a certain degree of control over granting permissions by allowing it to be done at runtime rather than during the installation of the app. Nevertheless, granting permissions at runtime does not solve the problem of malicious apps gaining access to sensitive personal information, as users may not have the knowledge required to grant only the permissions necessary for a particular app to operate. For example, a game app may request permissions to access user location data and to read the user's contact list. If granted, these privileges may lead to a privacy breach [8].

Inter-app communication channels may also pose a risk. For example, apps that appear benign on their own may become capable of performing a malicious task when working together (malware collusion). Such potentially dangerous apps may be hard to detect [9].

While earlier research in app security investigated how to distinguish between malicious and benign apps [10–14], more recent models and methods focus on evaluating the potential harmfulness of an app rather than classifying it as malicious or benign. For example, Feng et al. [4] used app permissions and descriptions to determine the “riskiness” of an app. Similarly, Wang et al. [15] proposed a framework that quantified app riskiness based on the permissions requested.

Alshehri et al. [7] developed a model that measures the security risk of Android apps based on the permissions that the user allows. The model (named PUREDroid) estimates the magnitude of the damage that might occur as a result of excessive permission-granting. For each app resident on the device, PUREDroid first creates two orthonormal state vectors representing the permissions the app has and has not requested. Then it determines the risk score of each app based on the number of times other known benign and malicious apps have also requested the permissions requested by the app. Higher-scoring apps are deemed potentially malicious. However, the accuracy of the model is relatively low. Benign apps that request excessive permissions will also receive a high risk score and will be deemed potentially malicious.

Also, based on app permission analysis, Rashidi et al. [16] proposed a risk assessment model named XDroid that monitors the resource usage of Android devices. Adopting a probabilistic approach (hidden Markov model), XDroid analyzes app behavior and performs an adaptive assessment of the apps residing on the device. Users select the resources they want monitored, and the system then alerts the user of suspicious activities related to the selected resources. As XDroid relies on user decisions, a lack of user expertise may negatively affect the choice of resources to be monitored, and thus the system's effectiveness.

The model proposed by Jing et al. [17] helps the user understand and mitigate the security risks that are associated with mobile apps and, in particular, with Android-based apps. The model (RiskMon) computes a risk score baseline that is derived from the runtime behavior of trusted apps and user expectations. The risk score baseline results are used to evaluate actual app behavior and generate cumulative app risk scores. RiskMon increases an app's baseline risk score every time an app attempts to access a sensitive or critical device resource. The model considers permission-protected resources only, assuming that these resources are reachable only if the respective permissions are granted. RiskMon reinforces resource protection by implementing automatic permission revocation that does not require user consent; this may have a detrimental effect on the effectiveness and efficiency of the MD user activities when using some services.

A comprehensive three-layer framework for assessing the risk posed by mobile apps was proposed by Li et al. [18]. Using a Bayesian graphical model, the system conducts

static, dynamic, and behavioral analyses to assess the risk that the app introduces to the mobile environment. The framework provides the user with information about apps that have lower risk profiles. However, the risk assessment is completed only after the app has been executed and the results of the analyses of all three layers have been combined into one final app risk score. Similarly, the models proposed by Kim et al. [5,19] consider the app's actual behavior, but the app needs to be executed to enable risk assessment. Such an approach leaves the MD dangerously vulnerable to the risk of compromise.

Baek et al. [20] proposed to measure the frequency of potential security events (e.g., financial loss or loss of private data) as an indicator of the riskiness of an app. They used a set of known benign and malicious apps and applied an unsupervised learning approach to create an app risk map. The device user can decide about using a particular app based on the plotted frequencies on the risk map. The model does not recommend or perform preventative actions when a risk is identified. Similarly, Kong et al. [21] propose a comprehensive risk evaluation approach that includes gathering information about the app from several sources. However, this model also relies on user judgment when determining the risk posed by an app.

More recent research investigates how to increase the accuracy of the prediction. For example, both Urooj et al. [22] and Boukhamla & Verma [23] propose ensemble machine learning (ML) models that work with a wide range of static features (including app permissions and intents). Panigrahi et al. [24] improved the feature selection process by adopting a high-performing nature-inspired approach to select the most suitable static features for the ML classification model (HyDroid). The DroidDetectMW model that was proposed by Taher et al. [25] uses both static and dynamic features to classify apps as either benign or malicious and utilizes a multi-class ML to determine the category that a malicious app belongs to. However, the effective implementation of these complex solutions may be hampered by the limited processing power of the MD.

The research reviewed recognizes the importance of assessing the risk posed by mobile apps residing on the MD. Three major challenges to accurate app risk evaluation can be highlighted:

1. How to reduce or eliminate dependency on inherently unreliable user input.
2. How to establish the riskiness of an app without having the need to execute it first.
3. How to increase the accuracy of the risk evaluation so that apps are not falsely categorized as risky.

In this research, we develop and evaluate a framework for app risk assessment that addresses these challenges. In addition to using app permissions as important risk characteristics, the framework considers app intents to capture data about app-to-app communication that may aid in collusive behavior. The framework includes an ensemble ML classification model and a probabilistic app risk assessment evaluator. It does not require user input or running the app that is being evaluated. Rather, an app is assigned a risk category based on the app's classification (benign or malicious) by the ML classifier, and the app's probabilistically estimated riskiness. Using this two-pronged approach mitigates the risk of falsely classifying an app as malicious.

The rest of the paper is organized as follows: Section 2 provides a description of the Android OS security mechanisms used in this research and the methods involved in data collection and analysis, and the proposed risk assessment framework and evaluation results are presented and discussed in Sections 3 and 4. Directions for further research are also outlined.

2. Materials and Methods

To ensure that the app activities do not affect other apps or the performance of the MD, each app operating on an Android OS platform uses its own sandbox [26]. This approach provides a secure environment since requests for communication with another app or for access to any sensitive resources are granted only if the app has the required permissions. Most apps require resources that are outside their sandboxes, which will be

possible if permission is granted. However, this user-controlled mechanism of granting privileges may expose the MD to potentially dangerous attacks [7]. For example, the user may unknowingly grant excessive permissions to a malware app that may use them to gain unauthorized access to sensitive personal information such as messages and call records [4]. Therefore, app features that were used for classification and risk evaluation in this study reflect two of the security mechanisms of the Android OS: the app permissions and the app intents as declared in the app's manifest file.

2.1. Android OS Security Mechanisms Used in This Study

App permissions protect the MD's functional capabilities. Once permission is granted, the app can invoke an API call for the functionality it needs. A complete list of all permissions required by an app is stored in the manifest file of the app. Most of the permissions used by the app belong to one of two major categories: normal permissions and dangerous permissions. Normal permissions are deemed to pose little or no risk to user privacy and security, and by default are granted to the app without informing the user [26]. On the other hand, for better protection, dangerous permissions (clustered into nine functional groups) are under user control, to be granted (or denied) at runtime (for Android OS 6.0 and above). However, a significant limitation of this model is the risk of granting privileges that exceed the scope of the functionality needed by the app. For example, an app might request the READ_PHONE_STATE permission; if granted, the app will have access to data such as the MD phone number, SIM (Subscriber Identity Module) serial card number, the SIM operator, and the IMEI (International Mobile Equipment Identity). Moreover, the app will also have access to all other functionalities in the same functional group such as the ability to make a phone call. The MD users may not always be aware of the consequences of granting excessive or dangerous permission [27].

In addition to normal and dangerous permissions, a third category, signature permission, comprises permissions that protect even more sensitive functional capabilities. One example is the WRITE_SETTINGS permission, which allows an app to modify system settings. For added protection, the app needs to send an "intent" before the setting change can be authorized.

Lastly, app intents are metadata components that are readily available in the manifest file of every Android app. An intent communicates the intention of an app to perform a certain action. It is a mechanism for coordinating different functional activities, including access control to the resources used by other apps within the MD. The purpose is to prevent an app from gaining direct access to other app data without having appropriate permissions. In this way, the intent mechanism controls what an app can do after it is installed, including intra-app and inter-app communication. The intent filter that is included in the app manifest file is also used to communicate the type of intent that an app can receive [24]. For example, an app may be designed to perform several different activities, with each activity on its own page and the user moving from one page to another. Appropriate intents will enable the passing of data from one activity to another, and from one activity component (e.g., an action button) to another component [28]. The intent mechanism provides an added layer of protection to sensitive MD resources and functional capabilities. Similarly to permissions, intents can be exploited by malware developers [11]. For example, two apps that appear benign on their own may, in fact, be designed to communicate with each other to perform a malicious task.

2.2. Study Sample and Datasets

To build the study's datasets, over 30,000 Android Package (APK) files were collected from the AndroZoo [29] and RmvDroid [30] repositories. The apps' APK files were screened with the antivirus engines of Virus Total, <https://virustotal.com> (accessed on 15 January 2022). The results were used to assign a label (benign or malicious) to each app. An app was deemed malicious if at least 15 of the VirusTotal antivirus engines flagged it as malicious, and benign if

none of the antivirus engines flagged it as malicious. This resulted in a sample of 28,306 apps comprising 9879 benign and 18,427 malicious apps.

The APK files were decompiled to obtain the manifest file of each app by applying the APK Easy tool, <https://apk-easy-tool.en.lo4d.com/windows> (accessed on 3 February 2022). Next, the permissions and intents of each app were extracted using custom-built software. It was established that across the collection of apps in the sample, a total of 132 unique permissions and 131 unique intents were used. Three study datasets were constructed as follows:

1. The dataset “Permissions” includes all apps in the sample. Each app is represented by a record that maps the use of the 132 unique permissions. The record comprises 132 binary input features (1 indicating that the respective permission is used by the app, and 0 indicating not used), and a label (1—malicious, 0—benign).
2. The dataset “Intents” was constructed similarly, with 131 binary input features indicating intent use and a label (benign or malicious).
3. The dataset “Hybrid” also includes all apps. Each app is represented by 263 features that map the combined permission and intent use, and a label (benign or malicious).

The data indicated that not all permissions and intents were highly used. Tables 1 and 2 show how the 25 most used normal and dangerous permissions and intents are used by the apps in the study sample.

Table 1. Permissions used by the apps in the study sample.

ID	Name	Type	Benign Apps (in %)	Malicious Apps (in %)
P1	WRITE_EXTERNAL_STORAGE	Dangerous	63.61	91.47
P2	READ_PHONE_STATE	Dangerous	25.84	96.52
P3	ACCESS_COARSE_LOCATION	Dangerous	24.95	68.2
P4	ACCESS_FINE_LOCATION	Dangerous	26.72	59.53
P5	GET_TASKS	Dangerous	6.49	50.17
P6	READ_EXTERNAL_STORAGE	Dangerous	30.58	33.42
P7	SYSTEM_ALERT_WINDOW	Dangerous	7.78	29.47
P8	READ_LOGS	Dangerous	1.85	30.57
P9	MOUNT_UNMOUNT_FILESYSTEMS	Dangerous	1.52	30.57
P10	CAMERA	Dangerous	19.34	19.7
P11	RECORD_AUDIO	Dangerous	8.31	20.18
P12	GET_ACCOUNTS	Dangerous	19.41	14.14
P13	CALL_PHONE	Dangerous	7.62	18.81
P14	WRITE_SETTINGS	Dangerous	5.7	16.5
P15	SEND_SMS	Dangerous	2.08	16.5
P16	INTERNET	Normal	98.8	99.89
P17	ACCESS_NETWORK_STATE	Normal	93.09	97.88
P18	ACCESS_WIFI_STATE	Normal	35.59	83.36
P19	WAKE_LOCK	Normal	58.29	45.16
P20	VIBRATE	Normal	34.04	50.92
P21	RECEIVE_BOOT_COMPLETED	Normal	23.3	38.22
P22	CHANGE_WIFI_STATE	Normal	4.92	31.42
P23	ACCESS_LOCATION_EXTRA_COMMANDS	Normal	1	22.65
P24	RESTART_PACKAGES	Normal	1	17.63
P25	MODIFY_AUDIO_SETTINGS	Normal	4.73	13.32

While both benign and malicious apps used some of the dangerous permissions, there were dangerous permissions that were used much more by malicious apps rather than by benign apps. One such dangerous permission was READ_PHONE_STATE, mentioned earlier. It was requested by 96.52% of the malicious apps in the sample. Another example is GET_TASK, which was used by 50.17% of the malicious apps. This permission enables access to all apps resident on a device. If obtained by a malicious app, it can lead to

significant compromise. In contrast, only a small number of the benign apps in the study sample requested these two permissions (6.49% and 25.84%, respectively).

Table 2. Intents used by apps in the study sample.

ID	Name	Benign Apps (in%)	Malicious Apps (in %)
I1	Action MAIN	99.86	98.59
I2	Category LAUNCHER	99.79	97.86
I3	Category DEFAULT	35.53	39.64
I4	Action BOOT COMPLETED	23.76	30.27
I5	Action PACKAGE ADDED	3.89	29.01
I6	Action VIEW	25.04	16.68
I7	Category BROWSABLE	22.72	14.19
I8	Action USER PRESENT	2.78	19.88
I9	Action PACKAGE REMOVED	1.98	14.15
I10	Category HOME	1.28	11.08
I11	Action SEARCH	4.66	2.24
I12	Action CREATE SHORTCUT	0.63	4.23
I13	Action MY PACKAGE REPLACED	6.58	0.16
I14	Action SEND	4.07	1.25
I15	Action PACKAGE REPLACED	2.08	2.25
I16	Action MEDIA MOUNTED	0.63	2.3
I17	Category LEANBACK LAUNCHER	3.83	0.36
I18	Action NEW OUTGOING CALL	0.69	2.02
I19	Action MEDIA BUTTON	3.21	0.44
I20	Action PACKAGE INSTALL	0.8	1.71
I21	Action SCREEN ON	0.26	1.44
I22	Category MONKEY	0.12	1.38
I23	Action TIMEZONE CHANGED	1.58	0.59
I24	Action SCREEN OFF	0.27	1.16
I25	Category INFO	0.87	0.81

Similarly, another five permissions (ACCESS_COARSE_LOCATION, ACCESS_FINE_LOCATION, READ_LOGS, MOUNT_UNMOUNT_FILESYSTEMS, SYSEM_ALERT_WINDOW) were used more frequently by the malicious apps rather than by the benign apps in the sample. In addition, both benign and malicious apps requested the dangerous permissions, WRITE_EXTERNAL_STORAGE and READ_EXTERNAL_STORAGE, with malicious apps still exhibiting more frequent use (91.47% vs. 63.61% and 33.42% vs. 30.58%, respectively). The dangerous permission request patterns that emerged indicated that app permissions may be used to make inferences about the potential of an app being malicious.

With respect to the usage pattern of intents in the sample, the two most often declared intents were Action_MAIN and Category_LAUNCHER, which were used both by benign and malicious apps with similar frequencies. However, there were some intents that showed relatively higher use by malicious apps compared to benign ones; for example, Action_USER_PRESENT (to know whether the MD is in use or idle) and Action_PACKAGE_ADDED (to add code to resident apps or to install a new app). While the percentage of malicious apps using these intents was not high in itself, their usage patterns provide additional insights into the app’s potential riskiness if considered in conjunction with the respective permission usage patterns.

Each of the study datasets was divided into two parts: 20% for testing and 80% for training and validation. The test datasets contained permissions and/or intents from 5661 apps (1995 benign and 3666 malicious apps). The training and validation sets contained permissions and/or intents from 22,645 apps. ML modeling was carried out using the Python programming language. The computer hardware included an Intel (R) Core (TM) i7-8700 CPU @3.20 GHz, 16 GB RAM, and a 500 GB hard disk drive.

3. Results

3.1. Risk Assessment Framework

The proposed risk assessment framework is shown in Figure 1. It includes an initial preparatory stage and an operational stage. At the initial stage, the ensemble ML classifier is trained and tested on input data that includes apps’ permission and intent usage indicators using a suitably constructed dataset. The trained classifier resides in the mobile cloud. The dangerous permissions used in calculating apps’ risk scores are also selected at the preparatory stage, and the respective dangerous permission risk scores are calculated using a probabilistic approach (as discussed in Section 3.3). In preparation for the operational stage, the ensemble ML classifier is downloaded and installed on the user’s MD along with the risk scores of the selected dangerous permissions and the software needed for the app risk score calculation and app risk evaluation.

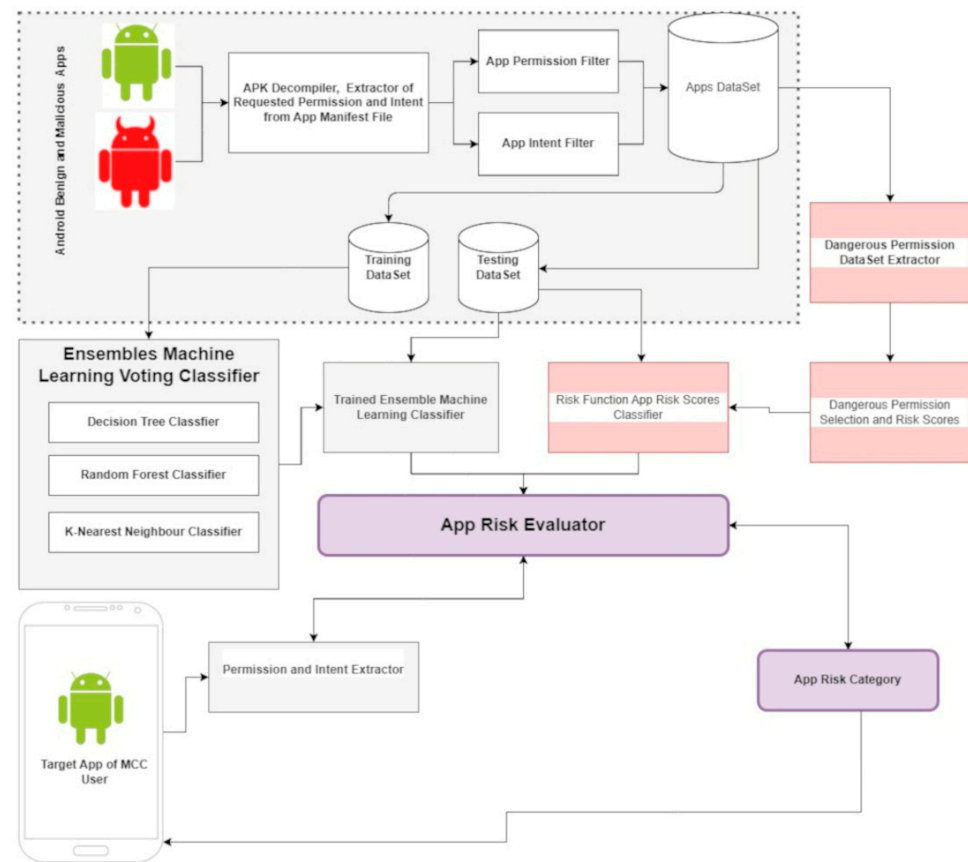


Figure 1. App risk assessment framework.

At the operational stage, the app risk evaluator extracts the permissions and the intents of the apps that are resident on the MD and feeds them to the trained ensemble ML classifier. The classifier checks each resident app and classifies it as either benign or malicious, while the app risk evaluator applies the risk function to calculate the app’s risk score. For each app, the risk function considers the permissions that belong to the set of the already selected dangerous permissions. Finally, the app risk evaluator assigns a risk category to each app.

The app risk evaluator aims to provide an accurate risk assessment at all times. Whenever a resident app is updated, the app risk evaluator automatically scans and re-evaluates the updated app because of the possibility of new permissions or intents to have been added to the new version. Furthermore, when the cloud ML model is retrained on a new dataset, the updated version is pushed to the MD and the app risk evaluator re-evaluates the resident apps.

3.2. Ensemble ML Model

A series of experiments were conducted to identify the most suitable algorithms for inclusion in an ensemble classifier to build an effective ML-based app classification model. The selected ML classification algorithms (shown in Table 3) have been used extensively in related prior research [31].

Table 3. Classification algorithms used in this study.

ID	Classification Algorithm
C1	Decision Tree
C2	Random Forest
C3	AdaBoost
C4	Naïve Bayes
C5	Stochastic Dual Coordinate Ascent
C6	Multi-Layer Perceptron
C7	K-Nearest Neighbor
C8	Linear Discriminant Analysis
C9	Logistic Regression
C10	Support Vector machine

Each algorithm was applied to each of the three study datasets. The outputs were compared to identify the best-performing classification algorithm based on the following confusion matrix:

1. True Positive (TP): The number of malicious apps classified correctly as malicious (in a given dataset).
2. True Negative (TN): The number of benign apps classified correctly as benign (in a given dataset).
3. False Negative (FN): The number of malicious apps classified wrongly as benign (in a given dataset).
4. False Positive (FP): The number of benign apps classified wrongly as malicious apps (in a given dataset).

The performance metrics used are listed below. The performance metric values obtained are shown in Table 4.

1. Classification Accuracy (CA): The percentage of all correctly classified apps (TP + TN) out of all apps in a given dataset (TP + TN + FP + FN).
2. Error Rate (ER): The percentage of all misclassified apps (FP + FN) out of all apps in a given dataset (TP + TN + FP + FN).
3. Precision (PR): The percentage of the correctly classified malicious apps (TP) out of all apps classified as malicious in a given dataset (TP + FP).
4. Recall (RC): The percentage of the correctly classified malicious apps (TP) out of all malicious apps in a given dataset (TP + FN).
5. False Positive Rate (FPR): The percentage of the wrongly classified malicious apps (FN) out of all benign apps in a given dataset (TN + FP).
6. False Negative Rate (FNR): The percentage of the wrongly classified benign apps out of all malicious apps in a given dataset (TP + FN).
7. False Alarm Rate (FAR): The mean of FPR and FNR.
8. F-Measure (FM): The harmonic mean of PR and RC.

Consistent with the data about intent usage by malicious and benign apps in the study sample, ML models using intents as the only features performed poorly compared to models using permissions only or those using both intents and permissions. Moreover, combining both permissions and intents as features consistently produced better results compared to using permissions only in terms of accuracy (CA) and false alarm rate (FAR). These results lend support to considering both app permissions and intents in the proposed risk assessment framework.

Table 4. Performance evaluation results.

Algorithm	Dataset	TP	FP	TN	FN	CA	ER	PR	RC	FM	FPR	FNR	FAR
C1	Permissions	3451	201	1774	235	92.3	7.7	94.5	93.62	94.06	10.18	6.38	8.28
	Intents	3455	1238	723	235	73.93	26.07	73.62	93.63	82.43	63.13	6.37	34.75
	Hybrid	3435	153	1842	231	93.22	6.78	95.74	93.7	94.71	7.67	6.3	6.99
C2	Permissions	3520	186	1789	166	93.78	6.22	94.98	95.5	95.24	9.42	4.5	6.96
	Intents	3478	1248	713	212	74.16	25.84	73.59	94.25	82.65	63.64	5.75	34.69
	Hybrid	3504	144	1851	162	94.59	5.41	96.05	95.58	95.82	7.22	4.42	5.82
C3	Permissions	3489	296	1679	197	91.29	8.71	92.18	94.66	93.4	14.99	5.34	10.17
	Intents	3428	1244	717	262	73.35	26.65	73.37	92.9	81.99	63.44	7.1	35.27
	Hybrid	3480	249	1746	186	92.32	7.68	93.32	94.93	94.12	12.48	5.07	8.78
C4	Permissions	1043	61	1914	2643	52.23	47.77	94.47	28.3	43.55	3.09	71.7	37.4
	Intents	1042	86	1875	2648	51.62	48.38	92.38	28.24	43.25	4.39	71.76	38.07
	Hybrid	1306	66	1929	2360	57.15	42.85	95.19	35.62	51.85	3.31	64.38	33.84
C5	Permissions	3543	333	1642	143	91.59	8.41	91.41	96.12	93.71	16.86	3.88	10.37
	Intent	3414	1251	710	276	72.98	27.02	73.18	92.52	81.72	63.79	7.48	35.64
	Hybrid	3488	248	1747	178	92.47	7.53	93.36	95.14	94.24	12.43	4.86	8.64
C6	Permissions	3471	202	1773	215	92.63	7.37	94.5	94.17	94.33	10.23	5.83	8.03
	Intent	3463	1261	700	227	73.67	26.33	73.31	93.85	82.32	64.3	6.15	35.23
	Hybrid	3461	164	1831	205	93.48	6.52	95.48	94.41	94.94	8.22	5.59	6.91
C7	Permissions	3526	243	1732	160	92.88	7.12	93.55	95.66	94.59	12.3	4.34	8.32
	Intent	2439	610	1351	1251	67.07	32.93	79.99	66.1	72.38	31.11	33.9	32.5
	Hybrid	3472	171	1824	194	93.55	6.45	95.31	94.71	95.01	8.57	5.29	6.93
C8	Permissions	3552	403	1572	134	90.51	9.49	89.81	96.36	92.97	20.41	3.64	12.02
	Intent	3460	1289	672	230	73.12	26.88	72.86	93.77	82	65.73	6.23	35.98
	Hybrid	3504	349	1646	162	90.97	9.03	90.94	95.58	93.2	17.49	4.42	10.96
C9	Permissions	3490	268	1707	196	91.8	8.2	92.87	94.68	93.77	13.57	5.32	9.44
	Intent	3455	1276	685	235	73.26	26.74	73.03	93.63	82.06	65.07	6.37	35.72
	Hybrid	3477	218	1777	189	92.81	7.19	94.1	94.84	94.47	10.93	5.16	8.04
C10	Permissions	3512	291	1684	174	91.79	8.21	92.35	95.28	93.79	14.73	4.72	9.73
	Intent	3453	1273	688	237	73.28	26.72	73.06	93.58	82.06	64.92	6.42	35.67
	Hybrid	3474	230	1765	192	92.55	7.45	93.79	94.76	94.27	11.53	5.24	8.38

The three best-performing classifiers in terms of CA, FAR, and false negative rate (FNR) were C2 (Random Forest), C1 (Decision Tree), and C7 (K-Nearest Neighbor). Applied to the hybrid dataset t, the top classifier (C2) achieved a CA of 94.59%, precision rate (PR) of 96.05%, and recall rate (RC) of 95.58%.

The outcomes of the tenfold cross-validation (Table 5) indicated that C1, C2, and C7 performed better than most, apart from C9 when using the Permissions dataset having the highest maximum CA overall (96.11%). However, C2 had the highest minimum CA (91.17%) overall. When applied to the hybrid dataset, C1 and C9 achieved the same maximum CA overall (95.41%); the highest minimum CA overall also belonged to C2 (90.46%). Considering these results, C1, C2, and C7 retained their position as the top-performing algorithms amongst the ten evaluated.

The top best-performing algorithms discussed above (Decision Tree, Random Forest, and K-Nearest Neighbor) were used to construct the ensemble ML classifier. The voting approach selected was “hard voting” (i.e., simple majority voting). Hard voting is reasonably effective for models working on discrete data such as the data used in this study. In addition, we aimed to build a relatively simple model that would not create computational overload when deployed on a low-resource MD.

The ensemble ML model used the app’s permissions and intents as features. It was trained and tested on the study’s Hybrid dataset described earlier. Table 6 presents the results along with the results obtained from applying each of the algorithms of the ensemble

to the same dataset. The comparison shows that the ensemble ML model performed better than the individual classifiers. It achieved a CA of 97.40%, error rate (ER) of 2.61%, PR of 97.92%, RC of 98.03%, and FAR of 2.87%.

Table 5. Tenfold cross-validation results.

Dataset	CA	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Permissions	Max	92.93	93.64	94.7	61.62	95.41	94.01	93.99	93.99	96.11	95.05
	Min	87.99	91.17	89.75	50.53	85.51	89.05	88.84	86.57	89.4	88.69
Intents	Max	74.82	75.18	76.24	42.2	73.05	75.27	73.14	72.08	76.24	75.53
	Min	69.5	69.86	69.15	37.46	57.6	70.57	65.37	67.08	69.96	69.96
Hybrid	Max	93.64	95.41	94.7	53	94.35	95.76	94.7	93.64	95.41	95.41
	Min	87.99	90.46	88.34	40.99	90.46	89.79	88.34	86.57	89.4	89.75

Table 6. C1, C2, C7, and ensemble ML model performance (using the Hybrid dataset).

	TP	FP	TN	FN	CA	ER	PR	RC	FM	FPR	FNR	FAR
C1	3435	153	1842	231	93.22	6.78	95.74	93.7	94.71	7.67	6.3	6.99
C2	3504	144	1851	162	94.59	5.41	96.05	95.58	95.82	7.22	4.42	5.82
C7	3472	171	1824	194	93.55	6.45	95.31	94.71	95.01	8.57	5.29	6.93
EMLM *	3583	76	1931	72	97.4	2.61	97.92	98.03	97.97	3.78	1.96	2.87

* EMLM—Ensemble ML model.

The ensemble ML model proposed and tested in this study outperformed the ML model considered in our earlier work [32]. The performance metrics achieved supported the use of the ensemble ML model discussed above as the “benign/malicious” app classifier in the risk assessment framework.

3.3. App Risk Evaluator

We now introduce our app risk evaluator. It determines the risk category of an app based on the classification output of the ML model, the estimated risk scores of a selected set of dangerous permissions, and the estimated risk score of the app.

3.3.1. Dangerous Permission Risk Estimation

The risk scores of the selected dangerous permissions are estimated as follows: let $P = \{P_1, P_2, P_3, \dots, P_n\}$ be a non-empty finite set containing n dangerous permissions P_i , where $1 \leq i \leq n$, and for any given set of apps X , let α_i be the ratio of the number of malicious apps in the set X using P_i to the number of all malicious apps in the set X . That is,

$$\alpha_i = \frac{\text{NumberOfMaliciousApps} \in \text{theDatasetUsing}P_i}{\text{NumberOfMaliciousApps} \in \text{theDataset}}$$

Similarly, let β_i be the ratio of the number of benign apps in the set X using P_i to the number of all benign apps in the set X . That is,

$$\beta_i = \frac{\text{NumberOfBenignApps} \in \text{theDatasetUsing}P_i}{\text{NumberOfBenignApps} \in \text{theDataset}}$$

For any i in the interval $[1, n]$, $0 \leq \alpha_i, \beta_i \leq 1$.

We calculate the risk score of a dangerous permission P_i that belongs to the set P as the value of the function $r(i)$ defined by

$$r(i) = \sqrt{\alpha_i - \alpha_i\beta_i} \tag{1}$$

The risk score of a dangerous permission P_i estimates the risk posed by the use of this permission based on its usage patterns across the set of apps X .

3.3.2. App Risk Score

The app risk score is estimated as follows. Let Z be a set of apps undergoing risk assessment. For any app z that belongs to Z and for each dangerous permission P_i from the set P , we define the function $\lambda(z, i)$ by

$$\lambda(z, i) = \begin{cases} 1 & \text{if the dangerous permission is used} \\ 0 & \text{if the dangerous permission is not used} \end{cases} \quad (2)$$

For any app z in Z , let $k(z)$ be the sum of the app's $\lambda(z, i)$ values. For any app z , the value of $k(z)$ is an integer where $0 \leq k \leq n$.

$$k(z) = \sum_{i=1}^n \lambda(z, i) \quad (3)$$

For any app z in Z , we calculate the app's risk score as the value of the function $R(z)$ defined by

$$R(z) = \frac{1}{k} \sum_{i=1}^n \lambda(z, i) r(i) \quad (4)$$

$R(z)$ is a probabilistic function with a value in the interval $[0, 1]$ (i.e., $0 \leq R(z) \leq 1$). It estimates the riskiness of the app z based on the combined risk potential of the dangerous permissions requested by z that belong to the set of dangerous permissions P . The app's risk score is used by the risk app evaluator to assign the app a risk category. The effectiveness of the approach of using the statistics of a suitable dataset to determine the riskiness of an Android app is supported by results reported in prior work [33,34].

3.3.3. App Risk Category

The app risk category is determined as follows: let Z be the set of all apps installed on a device, and for each app z that belongs to the set of the installed apps Z , the app evaluator calculates first the app's risk score $R(z)$ (i.e., the value of the risk function) using the risk scores $r(i)$ of a known set of n dangerous permissions $\{P_i\}$ where $i = 1, 2 \dots n$. Next, an app's risk category is determined as follows:

1. Low-Risk Apps

An app z that belongs to the set Z of all apps installed on a device is considered a low-risk app if, and only if, one of the following two conditions is satisfied:

- The ensemble ML model classifies the app as malicious and $0 \leq R(z) \leq t_1$ where t_1 is a predetermined threshold value in the interval $(0, 1)$.
- The ensemble ML model classifies the app as benign and the $0 \leq R(z) \leq t_2$ where t_2 is a predetermined threshold value in the interval $(0, 1)$ such that $t_1 < t_2$.

2. Medium-Risk Apps

An app z that belongs to the set Z of all apps installed on a device is considered a medium-risk app if, and only if, one of the following two conditions is satisfied:

- The ensemble ML model classifies the app as malicious and $R(z) < t_3$, where t_3 is a predetermined threshold value in the interval $(0, 1)$ such that $t_2 < t_3$.
- The ensemble ML model classifies the app as benign and $R(z) < t_4$, where t_4 is a predetermined threshold value in the interval $(0, 1)$ such that $t_3 < t_4$.

3. High-Risk Apps

An app z that belongs to the set Z of all apps installed in a device is considered a high-risk app if it does not meet any of the conditions above (i.e., the ensemble ML model classifies the app as malicious and $R(z) > t_3$ or the ensemble ML model classifies the app as benign and $R(z) > t_4$).

3.4. Risk Assessment Framework Validation

An instance of the framework was implemented to validate our approach. We used the trained ML classifier described in Section 3.2 and the precalculated risk scores of a selected set of known dangerous permissions. The selection of the known dangerous permissions and the permission risk score calculations are described below.

3.4.1. Known Dangerous Permissions

For the purposes of the experiment, we constructed the set of known dangerous permissions $PV = \{PVi, i = 1, 2, \dots, 15\}$ from the top 15 dangerous permissions P1, P2, P3...P15 listed in Table 1. These permissions were of particular interest, as they enable access to personal user data stored in the device.

The risk scores $r(i)$ ($i = 1, 2, \dots, 15$) of the selected dangerous permissions were calculated by applying the formula in Equation (1) (Section 3.3.1) to the apps in the set of apps X of the study sample (described in Section 2.2). The results are shown in Table 7.

Table 7. Risk scores of the selected dangerous permissions (permission set PV).

Permission	Calculated Risk Score
P2	0.8460
P3	0.7154
P5	0.6063
P1	0.5769
P9	0.5091
P8	0.4991
P4	0.4982
P6	0.4816
P10	0.3986
P15	0.3707
P12	0.3375
P14	0.2635
P7	0.2557
P13	0.2115
P11	0.1846

The dangerous permission READ_PHONE_STATE (P2) had the highest risk score (0.8460); as mentioned earlier, this permission gives access to the SIM card and the device details such as the IMEI and may enable targeted attacks. User privacy data could also be jeopardized if the dangerous permission ACCESS_COARSE_LOCATION (P3) was misused; this permission had the second-highest risk score (0.7154). The third-highest risk score value (0.6063) was that of the dangerous permission GET_TASK (P5), which provides access to the MD activity data.

The dangerous permission RECORD AUDIO (P11) had the lowest risk score (0.2115); when granted, this permission does not enable access to sensitive user data or device resources. The results were consistent with the functionalities associated with the top 15 dangerous permissions and their related permission groups.

3.4.2. Resident App Risk Assessment

To complete the framework validation experiment, we created an instance of the operational layer of the framework introduced in Section 3.1 (Figure 1) on an Android MD. The operational layer instance comprised a copy of the trained ML classifier described in Section 3.2 and purpose-built software for the app evaluator described in Section 3.3. The app evaluator used the set of known dangerous permissions and their risk scores, as described in the preceding section. The threshold values used to assess the risk of the apps resident on the MD were determined experimentally.

The validation sample of apps used was a set, ZV , of 20 apps downloaded from Google Play. By type, the apps belonged to ten different functional areas. These apps were

not part of the study datasets, as they were posted to the store after harvesting the apps that were used in the study sample. The apps were downloaded and installed on the MD. The apps’ APK files were checked by VirusTotal; none of the antivirus engines raised a flag for any of the apps, so all apps were deemed to be benign.

First, the trained ensemble ML classifier was applied to the validation sample and each app was classified as benign or malicious. Next, the app risk evaluator calculated a risk score for each app using the formulae in Equations (2)–(4) (Section 3.3.2). Finally, each app was categorized as high, medium, or low risk following the algorithm described in Section 3.3.3. The risk assessment results are shown in Table 8 for threshold values $t_1 = 0.25$, $t_2 = 0.50$, $t_3 = 0.65$, and $t_4 = 0.75$.

Table 8. Resident app risk score risk category, and dangerous permissions requested.

ID	Type	Name	Classification	Risk Score	Risk Category	Requested Dangerous Permissions from the Set PV
ZV1	Beauty	IPSY	Benign	0.4759	Low	P1, P3, P4, P6, P10, P11
ZV2	Beauty	Sephora	Benign	0.4888	Low	P1, P4, P6, P10
ZV3	Books	Read Extra	Benign	0.5293	Medium	P1, P6
ZV4	Books	GoodNovel	Benign	0	Low	None
ZV5	Business	Slack	Benign	0.4709	Low	P1, P2, P6, P10, P11, P12
ZV6	Business	Office Suite	Benign	0.6764	Medium	P1, P2, P5
ZV7	Dating	Tinder	Malicious	0.5385	Medium	P1, P2, P3, P4, P5, P6, P10, P11
ZV8	Dating	Zoosk	Benign	0.5506	Medium	P1, P2, P3, P4, P6, P10, P12
ZV9	Education	Moodle	Benign	0.5288	Medium	P1, P2, P3, P4, P6, P10, P11
ZV10	Education	LinkedIn Learning	Benign	0.5049	Medium	P1, P2, P3, P4, P6, P10, P11, P12
ZV11	Entertainment	Tubi	Benign	0.4496	Low	P1, P5, P6, P10, P11
ZV12	Entertainment	Xbox	Benign	0.655	Medium	P1, P2, P3, P6
ZV13	Game	Solitaire	Benign	0.5293	Medium	P1, P6
ZV14	Game	Bubble Shooter	Benign	0.4381	Low	P1, P6, P7
ZV15	Medical	FollowMyHealth	Benign	0.5769	Medium	P1
ZV16	Medical	Davis’s Drug Guide	Benign	0.5769	Medium	P1
ZV17	Shopping	Amazon	Malicious	0.4901	Medium	P1, P2, P3, P4, P5, P6, P7, P10, P11, P12
ZV18	Shopping	eBay	Benign	0.4455	Low	P1, P3, P4, P6, P10, P11, P14
ZV19	Social	Facebook	Benign	0.5341	Medium	P1, P3, P4, P6, P10
ZV20	Social	WhatsApp	Benign	0.5016	Medium	P1, P2, P3, P4, P5, P6, P10, P11, P12, P15

Across the set of the assessed apps, the number of dangerous permissions that the apps were requesting varied from zero to 10. Out of the 20 apps, seven apps were classified as low risk and 13 were classified as medium risk.

4. Discussion

As seen in Table 8, none of the 20 apps in ZV was categorized as a high-risk app. This was not unexpected, as all the apps in the validation sample were benign. The ML classifier correctly classified 18 apps as benign and wrongly classified two apps as malicious. We further analyzed these two apps (ZV7 and ZV17) to get an insight into this inaccurate prediction. It appeared that these apps used a relatively high number of dangerous permissions from the set PV (eight and ten, respectively). However, the apps’ risk scores were between 0.25 and 0.65, hence the app risk evaluator categorized them as medium-risk apps (rather than as high-risk apps).

An app’s risk category depends not so much on the number of known dangerous permissions requested by the app but on the particular risk scores of the dangerous permissions requested. For example, apps ZV11 and ZV19 requested the same number of dangerous permissions (five) but were categorized as low and medium risk, respectively. Indeed, ZV19 requested the second highest scoring dangerous permission, P3, while ZV11 requested the lowest-scoring permission, P11.

In another example, apps ZV15 and ZV16 each requested just one dangerous permission (P1). However, P1 is a high-risk permission, second only to P2. Both apps were categorized as medium risk. Only one app (ZV5) from the seven apps (ZV1, ZV2, ZV4, ZV5, ZV11, ZV14, ZV18) that were categorized as low risk used a highly dangerous permission

(P2). Generally, the medium-risk apps in the set *ZV* tended to use not just a high number of permissions but high-risk score permissions.

According to [35], the activities of malicious and unwanted apps have contributed significantly to the growth of the number of attacks on mobile devices. Installing fake apps on Google Play is one of the methods used by adversaries to obtain personal data about the MD user [36]. Indeed, it is pointed out in [37] that uploading malicious apps on Google Play has become quite common.

The proposed risk evaluation framework addresses the concerns of researchers and industry about the need to provide effective endpoint security, including on mobile devices, and the challenges identified in Section 1. First, the framework does not require user input. Second, it does not overload the MD, as it relies on the MCC environment to support its operations. The ML training and testing, selecting the set of dangerous permissions and calculating their scores, are processes that occur outside the MD. Third, the risk assessment is carried out without the need to run the app. In addition, the app's intent usage allows the ML classifier to learn the inter-app communication patterns. This enables the identification of apps that may attempt to evade detection using the existing "permission only" malware detection techniques. The combined use of these two types of app features adds to the reliability of the classification output as the model learns about new approaches to malware collusion (i.e., two apps that appear to be benign on their own but may be in fact communicating with each other to perform a malicious task).

Furthermore, the framework is flexible and adaptable to changes in the environment. As shown in Figure 1, the MDs that belong to the mobile cloud receive regular updates to maintain the app evaluator's accuracy. This allows necessary changes such as modifying the ML model or the training and testing dataset and changing the set of known dangerous permissions to be fed seamlessly into the risk assessment software operating at the MD level. The threshold values used to determine the risk category can also be varied. For example, they can be increased for MDs whose particular use is less tolerant of risk. These mechanisms address, to a degree, the issue of maintaining the sustainability of the model [22] given the changing threat and attack landscape.

The experimental results demonstrate the effectiveness of the proposed risk assessment approach and how combining the two risk assessment methods (i.e., the ML classifier and the app evaluator) can act as "check and balance," leading to an appropriate risk categorization of the resident apps. Although the validation sample was very small, the results obtained indicate that the risk assessment framework is realistic and reliable.

The approach proposed and evaluated in this study compares favorably to the latest research that focuses on app risk assessment using app features. Xiao et al. [38] determine the minimum set of permissions an app needs according to its functionality and use the difference between the minimum set and the set of actual permissions requested by the app to identify the app's "overprivileged" (unnecessary) permissions. An app is classified as risky if one or more of the unnecessary permissions are also risky (where the "risky" permissions are preidentified by an ML model). However, the model relies heavily on the assumption that Android app developers routinely include unnecessary permissions; this trend may not be persistent. Second, the method may not be able to classify correctly malicious apps that have been constructed through repackaging benign apps.

Yang et al. [39] also propose a risk assessment framework (PRADroid). It comprises a risk assessment matrix that considers the likelihood of an app being malicious (based on the app's permissions) and the severity of the consequences of data leakage. The computational engine includes an ensemble model of five ML classifiers and a code analyzer that examines the information flow path from the source API to the sink API. This framework has not been implemented as a prototype.

Dhalaria & Gandotra [40] have developed a web-based app where the user can upload an app's APK file to check the app's risk profile. The risk profile is determined by an artificial neural network model that uses the requested static permissions as app features. While

the reported classification accuracy is relatively high and the MD is not computationally overloaded, the app operates as a standalone one and requires significant user input.

5. Conclusions

This study's contribution is proposing an effective and feasible risk assessment framework for the protection of the data on mobile devices connected to the mobile cloud. First, to determine the risk scores of the selected dangerous permissions, the risk assessment model takes into account the relative frequency of permissions used by both benign and malicious apps in the training set. Permissions more likely to be used by malicious apps receive a higher risk score compared to permissions that are more likely to be used by benign apps. This ensures the model is not subject to the bias that may be introduced if only malicious apps were used. Second, rather than classifying an app as either benign or malicious, the model categorizes the app as low, medium, or high risk. The probabilistic evaluation combines the classification output of the ensemble ML model and the app risk score calculated by the app evaluator. Both approaches contribute to an effective reduction of the false alarm rate of the risk model.

Scalability is an important consideration that has not been addressed to date given the early-stage nature of the research and the fact that the tool is still, in essence, a minimum viable product (MVP). It will need to be given attention as the system is more widely deployed; there is nothing inherent in the way the tool has been built or implemented to suggest that it cannot scale effectively given the availability of sufficient computing resources. Another aspect to consider is developing a cross-platform solution that could work for different mobile operating systems. The challenge of this approach is how to reconcile the security mechanisms and policies of the different operating systems to build a uniform product. In the case of IOS, it may also be hard to obtain the apps needed to build a substantial dataset.

Directions for further research include the development of even more accurate ML models, experimental work to establish guidelines for setting threshold values, and extending the evaluation to include dynamic features such as API calls. Another avenue is the development and evaluation of an on-demand, cloud-based risk service model for comprehensive MD risk assessment.

Author Contributions: Conceptualization, N.O.O., K.P. and M.L.Y.; methodology, N.O.O., K.P., M.L.Y. and S.G.M.; software, N.O.O.; validation, N.O.O., K.P. and M.L.Y.; formal analysis, N.O.O., K.P. and M.L.Y.; investigation, N.O.O.; resources, N.O.O.; data curation, N.O.O.; writing—original draft preparation, N.O.O.; writing—review and editing, K.P., M.L.Y., N.O.O. and S.G.M.; visualization, N.O.O.; supervision, K.P., M.L.Y. and S.G.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The study data were derived from the following resources available in the public domain at the time of conducting the research: AndroZoo [29], RmvDroid [30]), and the Google Play store.

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

References

1. Sahi, L.; Sood, M.; Saini, S. Analysis and Evaluation of Mobile Cloud Computing: Service models, applications, and issues. In Proceedings of the 4th International Conference for Convergence in Technology (I2CT), Mangalore, India, 27–28 October 2018.
2. Nguyen, K.K.; Hoang, D.T.; Niyato, D.; Wang, P.; Nguyen, D.; Dutkiewicz, E. Cyberattack Detection in Mobile Cloud Computing: A Deep Learning Approach. In Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC), Barcelona, Spain, 15–18 April 2018.
3. Asnar, Y.; Hendradjaya, B. Confidentiality and Privacy Information Security Risk Assessment for Android-Based Mobile Devices. In Proceedings of the International Conference on Data and Software Engineering (ICoDSE), Yogyakarta, Indonesia, 25–26 November 2015.

4. Feng, Y.; Chen, L.; Zheng, A.; Gao, C.; Zheng, Z. Ac-net: Assessing the consistency of description and permission in Android apps. *IEEE Access* **2019**, *7*, 57829–57842. [[CrossRef](#)]
5. Kim, K.; Kim, J.; Ko, E.; Yi, J.H. Risk assessment scheme for mobile applications based on tree boosting. *IEEE Access* **2020**, *8*, 48503–48514. [[CrossRef](#)]
6. Alazab, M.; Alazab, M.; Shalaginov, A.; Mesleh, A.; Awajan, A. Intelligent mobile malware detection using permission requests and API calls. *Future Gene. Comput. Syst.* **2020**, *107*, 509–521. [[CrossRef](#)]
7. Alshehri, A.; Marcinek, P.; Alzahrani, A.; Alshahrani, H.; Fu, H. PUREDroid: Permission Usage and Risk Estimation for Android Applications. In Proceedings of the 3rd International Conference on Information Systems and Data Mining (ICISDM), Houston, TX, USA, 6–8 April 2019.
8. Bonné, B.; Peddinti, S.T.; Bilogrevic, I.; Taft, N. Exploring Decision Making with Android’s Runtime Permission Dialogs Using In-Context Surveys. In Proceedings of the Thirteenth Symposium on Usable Privacy and Security (SOUPS), Santa Clara, CA, USA, 12–14 July 2017.
9. Elish, K.O.; Cai, H.; Barton, D.; Yao, D.; Ryder, B.G. Identifying mobile inter-app communication risks. *IEEE Mob. Comput.* **2018**, *19*, 90–102. [[CrossRef](#)]
10. Idrees, F.; Rajarajan, M.; Conti, M.; Chen, T.M.; Rahulamathavan, Y. PIndroid: A novel Android malware detection system using ensemble learning methods. *Comput. Secur.* **2017**, *68*, 36–46. [[CrossRef](#)]
11. Feizollah, A.; Anuar, N.B.; Salleh, R.; Suarez-Tangil, G.; Furnell, S. AndroDialysis: Analysis of Android intent effectiveness in malware detection. *Comput. Secur.* **2017**, *65*, 121–134. [[CrossRef](#)]
12. Li, J.; Sun, L.; Yan, Q.; Li, Z.; Srisa-An, W.; Ye, H. Significant permission identification for machine-learning-based Android malware detection. *IEEE Trans. Industr. Inform.* **2018**, *14*, 3216–3225. [[CrossRef](#)]
13. Arp, D.; Spreitzenbarth, M.; Hubner, M.; Gascon, H.; Rieck, K.; Siemens, C. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In Proceedings of the 21st Annual Network and Distributed System Security Symposium, San Diego, CA, USA, 23–26 February 2014.
14. Saracino, A.; Sgandurra, D.; Dini, G.; Martinelli, F. Madam: Effective and efficient behavior-based Android malware detection and prevention. *IEEE Trans. Dependable Secur. Comput.* **2016**, *15*, 83–97. [[CrossRef](#)]
15. Wang, Y.; Zheng, J.; Sun, S.; Mukkamala, S. Quantitative Security Risk Assessment of Android Permissions and Applications. In Proceedings of the Data and Applications Security and Privacy XXVII: 27th Annual IFIP WG 11.3 Conference, Newark, NJ, USA, 15–17 July 2013.
16. Rashidi, B.; Fung, C.; Bertino, E. Android resource usage risk assessment using hidden Markov model and online learning. *Comput. Secur.* **2017**, *65*, 90–107. [[CrossRef](#)]
17. Jing, Y.; Ahn, G.J.; Zhao, Z.; Hu, H. RiskMon: Continuous and Automated Risk Assessment of Mobile Applications. In Proceedings of the 4th ACM Conference on Data and Application Security and Privacy, San Antonio, TX, USA, 3–9 March 2014.
18. Li, S.; Tryfonas, T.; Russell, G.; Andriotis, P. Risk assessment for mobile systems through a multilayered hierarchical Bayesian network. *IEEE Trans. Cybern.* **2016**, *46*, 1749–1759. [[CrossRef](#)]
19. Kim, H.; Cho, T.; Ahn, G.J.; Hyun Yi, J. Risk assessment of mobile applications based on machine learned malware dataset. *Multimed. Tools Appl.* **2018**, *77*, 5027–5042. [[CrossRef](#)]
20. Baek, H.; Joo, M.; Park, W.; You, Y.; Lee, K. Android Application Risk Indicator Based on Feature Analysis Utilizing Machine Learning. In Proceedings of the International Conference on Platform Technology and Service, Jeju, Republic of Korea, 28–30 January 2019.
21. Kong, D.; Cen, L.; Jin, H. AUTOREB: Automatically Understanding the Review-to- Behavior Fidelity in Android Applications. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, 12–16 October 2015.
22. Urooj, B.; Shah, M.A.; Maple, C.; Abbasi, M.K.; Riasat, S. Malware detection: A framework for reverse engineered Android applications through machine learning algorithms. *IEEE Access* **2022**, *10*, 89031–89050. [[CrossRef](#)]
23. Boukhamla, A.Z.E.; Verma, A. HyDroid: Android malware detection using network flow combined with permissions and intent filter. *Int. J. Mob. Commun.* **2023**, *22*, 70–91. [[CrossRef](#)]
24. Panigrahi, B.S.; Nagarajan, N.; Prasad, K.D.V.; Sathya; Salunkhe, S.S.; Kumar, P.D.; Kumar, M.A. Novel nature-inspired optimization approach-based SVM for identifying the Android malicious data. *Multimed. Tools Appl.* **2024**. [[CrossRef](#)]
25. Taher, F.; AlFandi, O.; Al-kfairy, M.; Al Hamadi, H.; Alrabae, S. DroidDetectMW: A Hybrid Intelligent Model for Android Malware Detection. *Appl. Sci.* **2023**, *13*, 7720. [[CrossRef](#)]
26. Rai, P.O. *Android Application Security Essentials*; Packt Publishing Ltd.: Birmingham, UK, 2013.
27. Alshehri, A.; Hewins, A.; McCulley, M.; Alshahrani, H.; Fu, H.; Zhu, Y. Risks behind device information permissions in Android OS. *Commun. Netw.* **2017**, *9*, 219–234. [[CrossRef](#)]
28. Bin Aftab, M.U.; Karim, W. *Learning Android Intents*; Packt Publishing Ltd.: Birmingham, UK, 2014.
29. Allix, K.; Bissyand’e, T.F.; Klein, J.; Le Traon, Y. Androzoo: Collecting Millions of Android Apps for the Research Community. In Proceedings of the IEEE/ACM 13th Working Conference on Mining Software Repositories, Austin, TX, USA, 14–15 May 2016.
30. Wang, H.; Si, J.; Li, H.; Guo, Y. RmvDroid: Towards a Reliable Android Malware Dataset with App Metadata. In Proceedings of the IEEE/ACM 16th International Conference on Mining Software Repositories, Montreal, QC, Canada, 26–27 May 2019.

31. Ogwara, N.O.; Petrova, K.; Yang, M.L. Towards the development of a cloud computing intrusion detection framework using an ensemble hybrid feature selection approach. *J. Comput. Netw. Commun.* **2022**, *2022*, 1–16. [[CrossRef](#)]
32. Ogwara, N.O.; Petrova, K.; Yang, M.L.B. MOBDroid: An Intelligent Malware Detection System for Improved Data Security in Mobile Cloud Computing Environments. In Proceedings of the 30th International Telecommunication Networks and Applications Conference, Melbourne, VIC, Australia, 25–27 November 2020.
33. Cen, L.; Gates, C.S.; Si, L.; Li, N. A probabilistic discriminative model for Android malware detection with decompiled source code. *IEEE Trans Dependable Secur. Comput.* **2014**, *12*, 400–412. [[CrossRef](#)]
34. Mat, S.R.T.; Ab Razak, M.F.; Kahar, M.N.M.; Arif, J.M.; Firdaus, A. A Bayesian probability model for Android malware detection. *ICT Express* **2022**, *8*, 424–431. [[CrossRef](#)]
35. Kivva, A. IT Threat Evolution in Q1 2024. Mobile Statistics. SECURELIST by Kaspersky. Available online: <https://securelist.com/it-threat-evolution-q1-2024-mobile-statistics/112750/> (accessed on 5 May 2024).
36. Cinar, A.C.; Kara, T.B. The current state and future of mobile security in the light of the recent mobile security threat reports. *Multimed. Tools Appl.* **2023**, *82*, 20269–20281. [[CrossRef](#)]
37. Nawshin, F.; Gad, R.; Unal, D.; Al-Ali, A.K.; Suganthan, P.N. Malware detection for mobile computing using secure and privacy-preserving machine learning approaches: A comprehensive survey. *Comput. Electr. Eng.* **2024**, *117*, 109233. [[CrossRef](#)]
38. Xiao, J.; Chen, S.; He, Q.; Feng, Z.; Xue, X. An Android application risk evaluation framework based on minimum permission set identification. *J. Systems Softw.* **2020**, *163*, 110533. [[CrossRef](#)]
39. Yang, Y.; Du, X.; Yang, Z. PRADroid: Privacy Risk Assessment for Android Applications. In Proceedings of the 5th IEEE International Conference on Cryptography, Security and Privacy, Zhuhai, China, 8–10 January 2021.
40. Dhalaria, M.; Gandotra, E. Risk detection of Android applications using static permissions. In *Advances in Data Computing, Communication and Security: Proceedings of I3CS2021*; Springer Nature: Singapore, 2022; pp. 591–600.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.