Scalable Computing: Practice and Experience, ISSN 1895-1767, http://www.scpe.org
© 2025 SCPE. Volume 26, Issues 4, pp. 1559–1576, DOI 10.12694/scpe.v26i4.4439

PRAZDROID: A NOVEL APPROACH TO RISK ASSESSMENT AND ZONING OF ANDROID APPLICATIONS BASED ON PERMISSIONS

ANURADHA DAHIYA*, SUKHDIP SINGH† AND GULSHAN SHRIVASTAVA‡

Abstract. The proliferation of Android apps has increased harmful apps that aim to influence user security, privacy, and device execution. Conventional detection techniques are becoming ineffective in identifying malicious applications as malware has enhanced its cognition and ingenuity and has reached a point where they are more impervious. Novel approaches based on machine learning have been provided to detect and classify malware threats. Still, the risk assessment of Android applications is significant for enhancing user trust and needs more attention. Permissions analysis is an effective way for risk assessment and behaviour study of Android apps because apps require permissions to access device functionality. In endorsement, this study proposes an approach (PRAZdroid) for risk assessment using permissions analysis. The proposed approach analyzed the M0droid dataset and computed five risk levels (Level 0 to Level 4). Statistical analysis is performed for risk levels and achieved 98 .07% classification accuracy with the Drebin and Anrdozoo datasets.

Key words: Android apps security, Permission analysis, User privacy, Risk assessment, Reverse engineering, Static malware analysis, Mobile security.

1. Introduction. The world of mobile devices is constantly changing as technology advances. Mobile users have been pleasured with increasing speed, storage capability, power, and availability of application services like games and online functioning. Malware attacks, specifically on Android devices, are rising with the growing favour of mobile devices. The most significant issues with Android are related to security, as it enriches competence with third-party software and open-source availability. Android apps are considerably optimistic to hackers as they are incredibly prevalent, with millions of users worldwide. These apps are evolving more insecure as hackers embed malicious code into them in intricate ways, making it complicated for security providers to identify and detect malicious apps. Android users can access applications from the official Play Store and third-party stores like ApkMirror and ApkPure. The official Android platform, Google Play Store, reported 2.59 million apps during the second quarter of 2023, an 8559.4 % increase from the launch of it [1].

Google enforces several security and privacy policies on apps listed in the Play Store to foster a vibrant app ecosystem and prevent users from engaging in malicious activity. However, challenges arise to balance security with developer freedom and user convenience, which eaves new paths for malpractice. Therefore, applications from the Google Play Store may not always be perfect; some apps from here have also been found to be malicious. Additionally, certain restrictions exist on accessing the Google Play Store in some places, such as China, Iran, and Cuba [2]. Alternatively, third-party app stores provide easy downloading and are operated by different organizations, such as contraption vendors and web service providers. The global diversity of these third-party stores has paved new paths for malware. According to PurpleSec cyber security report 2022, 98 % of mobile malware focuses on Android, and 99.9 % of observed mobile malware originated through third-party application stores [3].

Android uses the permission-based security model, allowing users to accept or reject app access to features and data through requested permissions. When users enthusiastically install an application, they stop thinking about the permission updates being asked by the application. They download the desired application and,

*Department of CSE, Deenbandhu Chhotu Ram University of Science and Technology, Murthal, India (anudahiya3973@gmail.com).

†Department of CSE, Deenbandhu Chhotu Ram University of Science and Technology, Murthal, India (sukhdeepsingh.cse@dcrustm.org).

‡School of Computer Science Engineering and Technology, Bennett University, Greater Noida, UP-201310, India (Corresponding author, gulshanstv@gmail.com)

when requested for installation, ignore everything else and initiate using it without considering the security consequences. Furthermore, the usual users ignore permissions as they do not have technical knowledge of permissions and their effects; it is challenging for them to make the right decision because both legitimate and malicious apps may ask for similar permissions [4]. Most of these apps ask for additional permissions and use user information without their awareness. An example of an additional requested permission is a calculator app that asks for the device's location and storage, demands access to the camera and internet, or loads unwanted packs. The apps that ask for more permissions than necessary can have the potential to transform from a benign to a harmful one.

However, Google has provided a multi-layered approach encompassing software, hardware, and user-enabled features for securing Android devices. Even so, Android does not have control of third-party sources, and downloading apps from these places can create the risk of malicious code or ads for users [5]. Simply detecting malicious apps is not enough in the present stature of malware progress; there is a need for a risk assessment system that enhances the capabilities of malware detection systems to inform users about the app's unreasonable permission requests and their impacts before app installation. So, this work analyzed the M0droid dataset [6] of 400 samples (200 benign and 200 malware) using reverse engineering and found 210 permissions. Examining these permissions using data parsing and set and map procedures results in 33 unique permissions used in malicious apps, 112 unique permissions used in benign apps, and 65 unique permissions used in both malware and benign apps. Based on these unique sets of permissions, five risk levels (level 0, level 1, level 2, level 3, level 4) are assessed.

1.1. Motivation. This work aims to obtain insights into the prediction of malicious Android apps using an ample permission analysis. The functionality of an app, which can be speculated from its description, is usually linked to the permissions it requires. Several privacy-invasive and malware applications have been observed to request more permissions than their alleged needs [7]. The present work took a malicious weather app, parsed its APK file, and observed that it requests permissions for "READ_CONTACTS" and "REQUEST_INSTALL_PACKAGES". These permissions have no impact on the app's intended functionality but may compromise users' privacy. Existing works have preferred binary classification using important prophets. However, risk assessment has been overlooked, encouraging the user to lower the unwilling installation of malicious applications. Risk assessment specifies the significant measures to inform the user in the permissive mode and ensures trust.

The application of rule-based models and transparency of predictions is limited to a few studies such as Karim et al. [8] adopted an methodology based on association rule mining. The present work leverages data sources to identify predictive patterns of malicious Android apps. It departs from the conventional use of predictive regression models by exploiting the potential of rule-discovery techniques to generate patterns that link malicious permission presence with risk factors. Users can figure out the reasons behind the predictions as the suggested work uses a collection of rules that describe data and assist in enhanced prediction capability.

The remanent sections of this paper are structured as follows: Section II summarises relevant literature. Section III delves into an elaborate discussion of the proposed method. Section IV looks at the exploration & results evaluation of the proposed method. Section V covers the analytical discussion on risk factors. Finally, Section VI infers the work & provides suggestions for further research.

2. Related work. An efficient risk assessment can generate a risk-based prioritized list of untrusted input apps. This list helps both users and innovators. Users can install and use low-risk apps, and innovators can use it to select high-risk apps for further malware analysis. This section involves the miscellaneous practices introduced in the literature to identify malicious behaviour apps and explore allied risk. Scientific literature on Android applications risk analysis is confined and predominantly concentrates on permissions. Xiao et al.[9] suggested permission analysis to recognize the difference between the minimum permissions needed for an app to perform its work and the requested permissions of that app. This work combined collaborative filtering and static analysis to find additional permission requests for an app and, based on additional requests, evaluate the associated risk of the app.

Deypir and Horri [10] provided a metric that used instances of previously known malicious and non-malicious apps to estimate the risk of unknown apps instead of using features such as permissions, intents, etc. This approach presents previously known samples in a high-dimensional feature space. It computes the associated

risk of an unknown app using its distances to known malicious and non-malicious app instances. The Euclidean distance measure has been used here to estimate apps' effectual security risk score. Most device users see that graphical indications that present the summary of risk or safety scores work better for notifying them than textual information on permissions. As Dhalaria and Gandotra [11] provided a risk detector for Android apps using permissions features and an artificial neural network that identifies risk based on the probability of benign and malware data samples. This work also designed a graphical user interface for uploading and testing the app's behaviour.

An adequate approach encourages users to select safe apps from the Android App Stores when the stores contain different apps for the same functionality with variant risk scores. Sharma and Gupta [12] perused Android apps using permissions analysis to determine the associated risks. This work initially analyzed the M0Droid dataset samples permissions requests through reversing and achieved 165 permissions with usage rates in malware and benign apps. Then, it compared the permissions from both types and quantified the risk into four factors. The work also performed a statistical evaluation using ANOVA and t-tests to show the mutual exclusiveness of risk factors. AlOmari et al.[13] looked at the effectiveness of several machine-learning algorithms in identifying malware for Android devices. Their approach used PCA, normalized the numerical features, and employed the Synthetic Minority Oversampling Technique (SMOTE) to accomplish higher accuracy. This work identified Android malware and classified them into five categories: benign, adware, SMS malware, banking malware, and mobile riskware using the Light Gradient Boosting Model.

Feature reduction assists in deciding the most pertinent features and enhances the machine learning model results by allowing better distinction between benign and malicious apps. Sharma and Arora [14] provided an approach that integrates intents & permissions. Normal & malware apps may use the same feature patterns, but this approach ranks intents & permissions using a Chi-square test based on frequency to find distinctive features. This work applied various deep learning & machine learning classifiers on the combined ranked permissions and intents and achieved 98.49 % recognition accuracy. Upadhayay et al.[15] suggested fraudulent activity recognition using permissions ranking & network traffic features. This work ranked commonly used permissions in benign & malware apps, then removed the lower-ranked permissions using several thresholds. It provided impressive results by applying machine learning models on a hybrid feature vector of the best permissions and network traffic features.

Saracino et al.[16] provided a cross-layer classification model based on machine learning using hybrid features, system calls, API, user activity, SMS, and application metadata. This work achieved 96.6 % accuracy with the Genome, VirusShare, and Contagio datasets. The authors also pleaded that the presented work presents low-performance overhead and limited battery consumption. Malleswari et al.[17] suggested an approach to increase user awareness before allowing any permission. This work considered individual evaluation of the permissions, negotiation of permissions, & the relative significance of permissions. The work recommended a risk score derived with the assistance of fuzzy AHP based on permissions asked by the application.

The above-discussed approaches are a slight part of suggested and implemented detection methods; with constantly updating technology, attacks are expanding rapidly and elongating the urge for new approaches. For example, malware applications can auto-root themselves on devices and install other applications without the user's consent. This indicates the need for improved permission-based Android security methods to alert users to malicious activity.

3. Proposed method. The main goal of the presented work is to extract prediction conventions for Android apps to identify the risk level while installing them. The presented methodology is named PRAZdroid as Permission-based Risk Analysis and Zoning of Android apps, which consists of four parts collection of data, data pre-processing and aggregation, data analysis, and prediction, as shown in Fig. 3.1.

Various apps are collected from diverse sources during the data collection phase. These sources have been identified with the help of relevant literature expressed by Dahiya et al.[18]. The data aggregation and pre-processing phase has extracted permissions from app APK files and constructed permission groups based on their use in apps. Data analysis involves rule discovery algorithms to extract hidden patterns from the labelled training dataset. These patterns reveal the relationship between the occurrence of classes and which combination of aspects led to a lower or higher risk of malware occurrence. In the last prediction phase, an assessment is executed to evaluate the performance of predictions, and the end users are notified about the

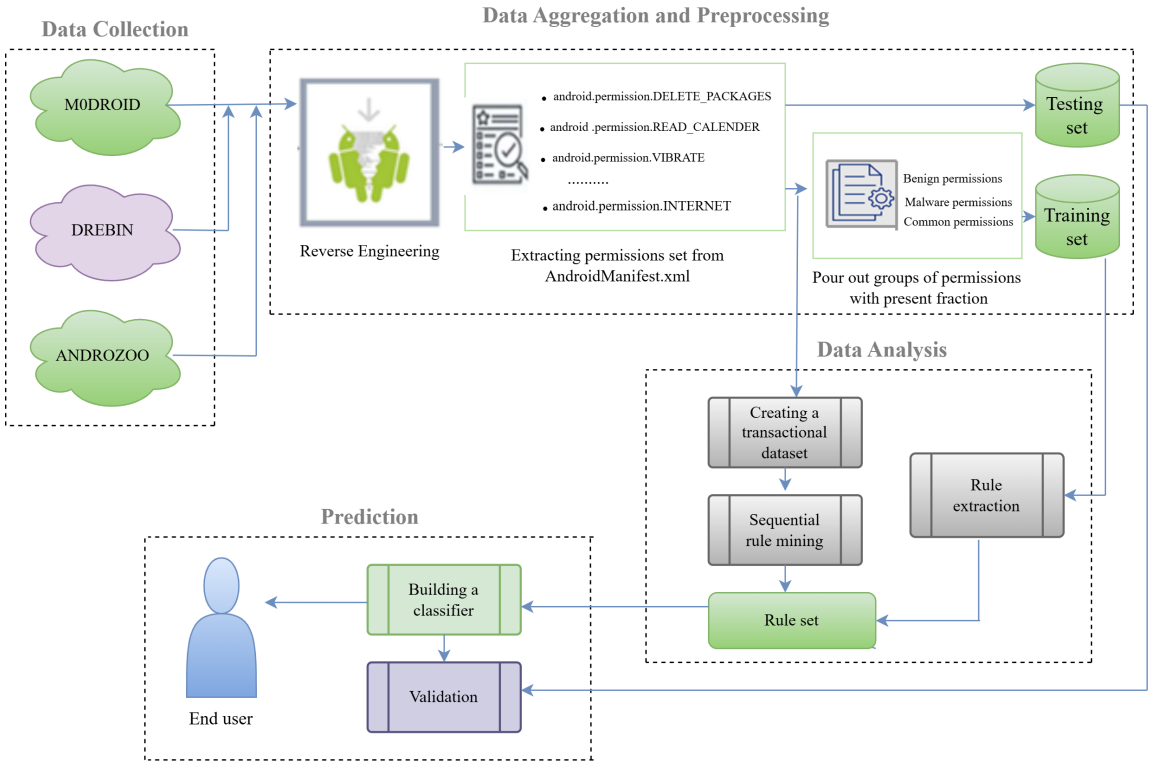

Fig. 3.1: PRAZdroid - Research Methodology

apps' associated risks.

3.1. Data collection. The data has been downloaded from three different sources, and an overview of them is presented in Table 3.1. These data sources are freely available online, requiring permission from the respective research committee. Downloading AndroZoo data samples requires the API key of the authorized user and the SHA256 value of the required app. Similarly, Drebin requires user login details of authorized users. The present work has been analyzed with a substantial collection of 400 M0droid, 1350 AndroZoo, and 1350 Drebin app samples.

The permissions of apps from the M0droid dataset have been explored for training purpose. For testing Drebin and Androzoo data collection have been taken, as Fig. 3.2 visualizes these sources and summarises the information they provide about apps. Drebin contributes sample files of malware apps and feature vectors of a large collection of malware and benign apps; with these sample files and feature vector permissions of apps can be identified. Current work collected 1350 malware sample files from Drebin. Androzoo bestows a large collection of apps in the form of apk files with information about the Virustotal detection mark, size of apk, scan date, source of that apk, etc. It also provides app metadata in the form of manifest permission lists, etc. From metadata and apk of apps, permissions can be extracted. The apps with a virus total detection value of zero are considered benign apps; similar 1350 benign apps are taken from Androzoo.

3.2. Data aggregation and pre-processing. Android has followed a permission policy with predefined permissions to perform specific activities. Any program can ask for the necessary authorizations. The required permissions are specified by Android programs in the Android Manifest. In their manifest file, applications must specify which permissions they want or need [21]. Android permissions control necessary access to application data. Without the required permissions, data stored on the computer cannot be accessed. An Android app is distributed through a packaged APK (Android Package Kit), which bundles all the essential resources to

Table 3.1: Data sources

Source	Description
M0droid [6]	This dataset is bundled with an adequate approach for Android malware recognition by leveraging behavioural analysis. It generated signatures for apps based on their system call requests, normalized the generated signatures using z-score and median algorithms, and identified malware by comparing behavioural signatures with blacklist signatures. The dataset contains 400 samples of apps, with 200 malware and 200 benign apps.
Drebin [19]	It is widely used for Android malware analysis and detection research, including many apps from benign and various malware families, FakeInstaller, DroidKungFu, Opfake, Kmin, Plankton, etc. This dataset is bundled with an adequate approach that integrates static analysis & machine learning. It provides feature vectors for 1,29,013 samples, of which 1,23,453 are benign and 5,560 malware. It also provides 5,560 malware sample files from 179 families and family labels for these files.
Androzoo [20]	It is a massive dataset of Android apps that serves as a worthwhile resource for researchers, application analysis, and security assembled from various sources, including Anzhi, Appchina, PlayDrone, Google Play Store, Slideme, VirusShare, etc. It is constantly updated to reflect the evolving landscape of Android apps and labels them malicious or harmless based on analysis by various antivirus products. Ample metadata, including VirusTotal reports, static code analysis, manifest permissions, and behavioural analysis accompany each app.

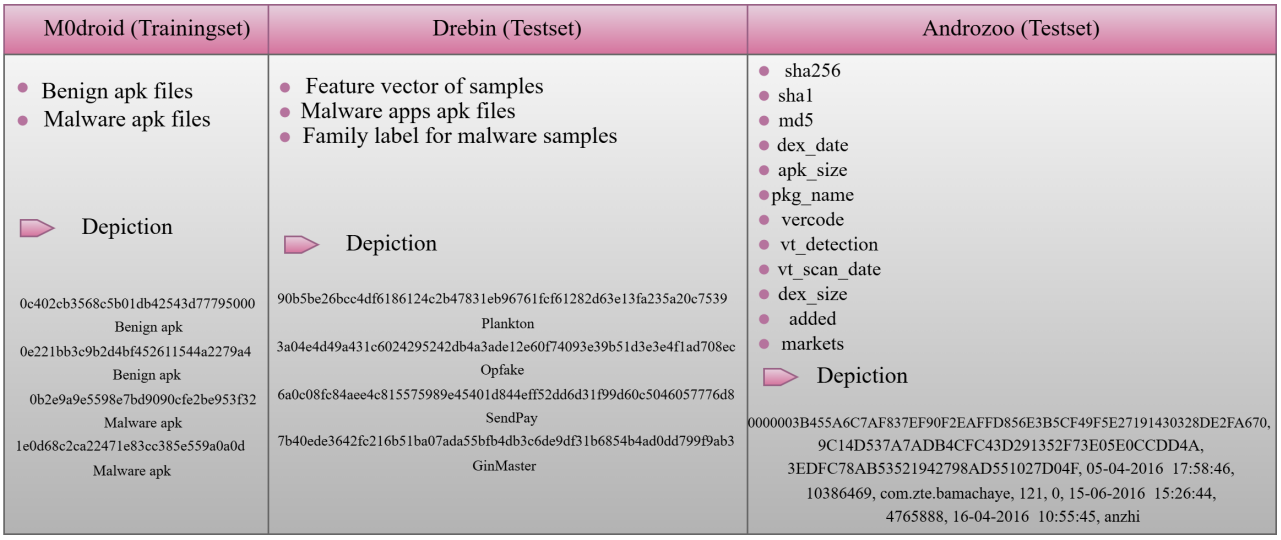

Fig. 3.2: Data sources essence

operate the app. AndroidManifest.xml file is the part of the APK that describes important details of the application, such as the Application name, Required permissions, Version information, Package name, etc [22].

Firstly, the present work performed reverse engineering using ApkTool [23] as described in Algorithm 1 to analyze the permissions sought by Android apps in the M0droid dataset. ApkTool is an emphatic reverse engineering tool for Android applications, which has been used here to extract manifest files. Fig. 3.3 shows the manifest file of a benign app from the collection that discloses the list of required permissions and provides valuable particulars about the app's possible behaviour & data access. These extracted manifest files of each malware and benign app have been read and parsed using the Python library BeautifulSoup with an 'XML' parser to pull the respective permissions of apps. All elements that start with `<uses-permission>` have been searched, as shown in Fig. 3.4 these specify the permission. The value of the 'android:name' distinctive from each `<uses-permission>` element has been extracted. Permissions of benign and malware apps have been collected in respective benign and malware archives with their computation of uses. The similarity of these archives has been measured using statistical measures Jaccard similarity, which is calculated as the entirety of

Algorithm 1 Manifest Extraction from APKs

Input:
$M = \{M_1, M_2, M_3, \ldots, M_n\}$ // APK of Android malware applications
$B = \{B_1, B_2, B_3, \ldots, B_n\}$ // APK of Android benign applications
Output:
$ManifestB = \{ManifestB_1, ManifestB_2, ManifestB_3, \ldots, ManifestB_n\}$ // AndroidManifest.xml for benign apps
$ManifestM = \{ManifestM_1, ManifestM_2, ManifestM_3, \ldots, ManifestM_n\}$ // AndroidManifest.xml for malware apps
procedure :
Initialize empty lists for Manifest files:
$ManifestB \leftarrow emptylist$
$ManifestM \leftarrow emptylist$
function EXTRACT_MANIFEST(apk)
Run ApkTool(apk) to extract AndroidManifest.xml
if ApkTool returns SUCCESS **then**
return extracted AndroidManifest.xml
else
return "Extraction Failed"
end if
end function
for each APK B_i in benign applications B **do**
$ManifestB_i \leftarrow$ extract_manifest(B_i) // Extract manifest of benign APK
Append $ManifestB_i$ to $ManifestB$
end for
for each APK M_i in malware applications M **do**
$ManifestM_i \leftarrow$ extract_manifest(M_i) // Extract manifest of malware APK
Append $ManifestM_i$ to $ManifestM$
end for
return $ManifestB, ManifestM$ // Return the lists of manifests for benign and malware apps
end procedure

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="47" android:versionName="0.9.7.3" package="eu.thedarken.sdm">
    <uses-sdk android:minSdkVersion="4"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <application android:theme="@style/Theme.NoTitleBar" android:label="@string/app_name" android:icon="@drawable/icon">
        <activity android:label="@string/app_name" android:name=".SDMmain" android:configChanges="locale|keyboardHidden|orientation">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity android:theme="@style/Theme.NoTitleBar" android:label="Corpse finder(Samsung)" android:name="SDMstartpage"/>
        <activity android:theme="@style/Theme.NoTitleBar" android:label="Explorer" android:name="SDMexplorer"/>
        <activity android:theme="@style/Theme.NoTitleBar" android:label="Searcher" android:name="SDMsearch"/>
        <activity android:theme="@style/Theme.NoTitleBar" android:label="Corpse finder(Android)" android:name="SDMcorpsefinder"/>
        <activity android:theme="@style/Theme.NoTitleBar" android:label="Clean system" android:name="SDMcleanSystem"/>
        <activity android:theme="@style/Theme.NoTitleBar" android:label="Clean Apps" android:name="SDMcleanApps"/>
        <activity android:theme="@style/Theme.NoTitleBar" android:label="Duplicates" android:name="SDMduplicates"/>
        <activity android:theme="@style/Theme.NoTitleBar" android:label="Vacuum" android:name="SDMvacuum"/>
        <activity android:theme="@style/Theme.NoTitleBar" android:label="Misc" android:name="SDMmisc"/>
        <activity android:theme="@style/Theme.NoTitleBar" android:label="Preferences" android:name="Preferences"/>
    </application>
    <supports-screens android:anyDensity="true" android:smallScreens="true" android:normalScreens="true" android:largeScreens="true"/>
</manifest>
```

Fig. 3.3: Sample of manifest file

the intersection divided by the entirety of their union, as shown in Eq.(3.1).

$$\text{Jaccard Similarity}(B, M) = |B \cap M|/|B \cup M| \tag{3.1}$$

The calculated similarity of 0.30 showed that 30% of permissions have been commonly used in both malware & benign applications. This measure shows that the commonly used permissions are significant and require

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
```

Fig. 3.4: Requested permissions

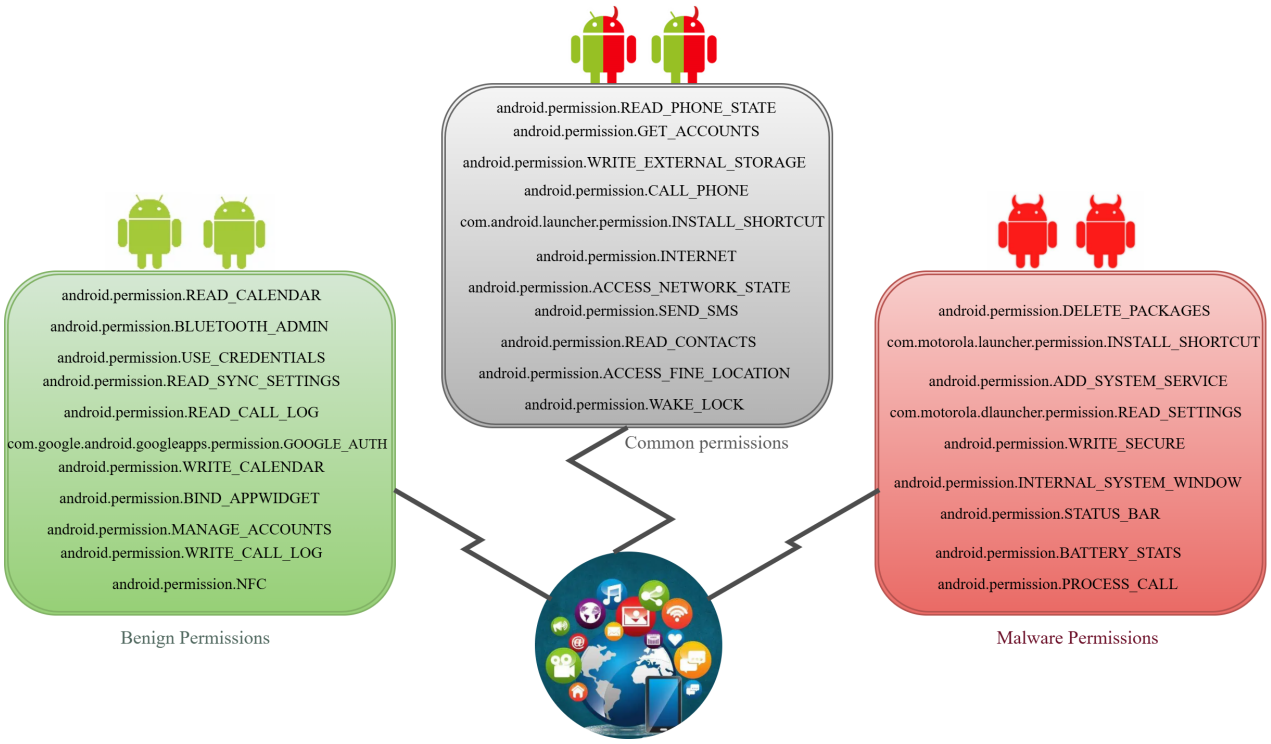

Fig. 3.5: Top-requested permissions of Android apps

regard for the soundest analysis, so three lists have been prepared as malware, benign, and common permissions with a presence fraction. Fig. 3.5 shows an overview of the top permissions sought by benign & malicious apps. This data represents the permissions that are sought exclusively by benign apps (benign permissions), those solely demanded by malicious apps (malware permissions), and permissions that are commonly sought by both types of apps (common permissions).

3.3. Data analysis. These sets of permissions are dissected with JRip (RIPPER) and PrefixSpan (Prefix-projected Sequential Pattern Mining) models to discover hidden rules for indicating the risk of malware occurrence. These rules are extracted in the "IF-THEN" form as demonstrated in Eq.(3.2). In this statement, P is the premise, and Q is the consequence. This means that Q is correlated with P because if P is satisfied, then Q is true.

$$IF\ (P \text{ is } X)\ THEN\ (Q \text{ is } Y) \tag{3.2}$$

Relating Eq.(3.2) to permission-based malware analysis problems, it can be penned as an occurrence of risky permission assets for the malware specified in Eq.(3.3).

$$IF\ (Risky\ permission\ is\ Present)\ THEN\ (App\ is\ Malicious) \tag{3.3}$$

A rule consists of various interconnected elements and their coverage. These extracted rules are amalgamated to form the rule set for the final classifier. The training set is prepared with five attributes i.e.,

permission, benign frequency, malware frequency, malware ratio, and class. The malware ratio is fixed for only benign and only malware permissions, while for common permissions, it has been computed with benign and malware frequency as specified in Eq.(3.4).

$$Malware\ ratio = \frac{Malware\ frequency\ of\ permission}{Benign\ frequency\ of\ permission} \tag{3.4}$$

With the prepared training set, a supervised data mining algorithm JRip has been trained using the WEKA tool [24] to induce rules. JRip [25] is a rule-based classifier that acquires rules explicitly from the training data in IF-THEN statements form for making predictions. It generates rules through four steps i.e., growth, pruning, optimization, and selection. Growth starts generating one rule by adding attributes to the rule until the stopping criteria are met. Pruning shortens each rule by removing redundancy. Optimization tries to generate more rules from the ruleset, and the selection phase selects effective rules. Repeated Incremental Pruning to reduce error, Interpretability of rules, efficiency with large datasets, and stability with multiclass problems make JRip a satisfactory choice.

The Explanatory variables are discretized into Negligible(N), Minor(M), Moderate(MD), Likely(L), and Very likely(VL) based on histograms of these. The model has been trained using a 5-fold cross-validation. Precision, Accuracy, F-measure, & Recall metrics are produced in each round. Accuracy is a common indicator used to evaluate the classification effectiveness of a model in terms of the overall proportion of correct predictions. Recall determines the proportion of factual positive instances correctly identified. Precision reflects the proportion of accurate positive prophecies. F-measure combines recall & precision calculated as the harmonic mean of these. The influence of rules is quantified using support and confidence criteria. Support directs to the frequency of occurrences in the dataset that support a particular rule, and confidence refers to the frequency with which a rule statement is true.

An additional analysis has been performed to understand the sequential relationship between permissions acquired by apps. These findings can be helpful in better understanding the occurrence of malware. The transactional dataset for both malware and benign apps has been prepared as shown in Table 3.2. The malware transactional dataset includes each malicious app permissions request as a transaction. Similarly, the benign transactional dataset includes each benign app permissions request as a transaction. These datasets are used as input for the popular sequential pattern-mining algorithm PrefixSpan [26], which leads to only one pass through the data sequence to identify frequent items. PrefixSpan reduces search space and improves efficiency by promoting a divide-and-conquer process instead of a pattern-growth process that directly projects the database based on frequent prefixes. The outcomes of this analysis revealed patterns of permissions used that contribute to preparing the rule set for the final classifier.

3.4. Prediction. The observations of prefixspan and rules extracted from JRip are converged to construct the ruleset for defining the final classifier. The rule set creation and risk prediction process is shown in Fig. 3.6 The frequent sub-sequences of permissions used in malware & benign apps and their behavioural analysis facts extracted using rule predictor provide a pleasing base for ruleset creation. The risk level of an app is evaluated by examining its requested permissions. Each requested permission is compared with the predefined ruleset, and a corresponding score is given based on the matching. If permission matches the critical group rules, assign its score value as critical. Similarly, the matching of permissions with neutral, high, and low group rules is assigned with neutral, high, and low scores. Permissions that do not match any rules are assigned a default score.

Now, the risk level of the app is estimated based on frequencies of scores assigned to its requested permissions. A positive critical score classifies the app as a level 4 risk, and the frequency of the critical score is the risk value that indicates the severity of the risk at this level. Apps with naught critical scores and a positive high score are classified as level 3 risk. The frequency of the high score shows risk severity at this level. The app with naught critical, high scores and a positive neutral score is classified as level 2 risk. The frequency of the neutral score shows the risk severity of the app at this level. Similarly, naught critical, high, and neutral scores with a positive low score classify an app as a level 1 risk, and the frequency of the low score reveals risk severity. An app with all other scores naught and a positive default score is believed to be a normal app without malicious activity, classified as level 0. The default score frequency reveals the app's potency.

Table 3.2: Sample of the transactional datasets

Malware transactional dataset	<”android.permission.INTERNET”,”android.permission.READ_PHONE_STATE”,”android.permission.READ_CONTACTS”>
	<”android.permission.CALL_PHONE”,”android.permission.INTERNET”,”android.permission.READ_PHONE_STATE”,”android.permission.READ_CONTACTS”,”android.permission.ACCESS_NETWORK_STATE”>
	<”android.permission.READ_PHONE_STATE”,”android.permission.ACCESS_NETWORK_STATE”,”android.permission.SEND_SMS”,”android.permission.INTERNET”,”android.permission.WRITE_EXTERNAL_STORAGE”,”android.permission.INSTALL_PACKAGES”,”android.permission.DELETE_PACKAGES”>
	<”android.permission.ACCESS_WIFI_STATE”,”android.permission.INTERNET”,”android.permission.READ_PHONE_STATE”,”android.permission.WRITE_EXTERNAL_STORAGE”,”android.permission.ACCESS_NETWORK_STATE”>
	<”android.permission.WRITE_EXTERNAL_STORAGE”,”android.permission.INTERNET”,”android.permission.READ_PHONE_STATE”,”android.permission.READ_SMS”,”android.permission.SEND_SMS”,”com.software.application.permission.C2D_MESSAGE”,”com.google.android.c2dm.permission.RECEIVE”,”android.permission.RECEIVE_SMS”,”android.permission.WAKE_LOCK”>
	.....
Benign transactional dataset	<”android.permission.INTERNET”,”android.permission.VIBRATE”,”android.permission.ACCESS_COARSE_LOCATION”,”android.permission.READ_CALENDAR”,”android.permission.WRITE_EXTERNAL_STORAGE”>
	<”android.permission.INTERNET”,”android.permission.ACCESS_WIFI_STATE”,”android.permission.ACCESS_NETWORK_STATE”,”android.permission.WAKE_LOCK”,”android.permission.WRITE_EXTERNAL_STORAGE”,”android.permission.RECEIVE_BOOT_COMPLETED”,”com.android.vending.CHECK_LICENSE”>
	<”android.permission.INTERNET”,”android.permission.ACCESS_NETWORK_STATE”>
	<”android.permission.INTERNET”,”android.permission.ACCESS_NETWORK_STATE”,”android.permission.RECEIVE_BOOT_COMPLETED”,”android.permission.GET_ACCOUNTS”,”android.permission.WAKE_LOCK”,”com.mobiusx.live4dresults.permission.C2D_MESSAGE”,”com.google.android.c2dm.permission.RECEIVE”,”android.permission.VIBRATE”>
	<”android.permission.RECORD_AUDIO”,”android.permission.VIBRATE”,”android.permission.WAKE_LOCK”,”android.permission.READ_PHONE_STATE”,”android.permission.CAMERA”,”android.permission.WRITE_EXTERNAL_STORAGE”,”android.permission.INTERNET”,”android.permission.SEND_SMS”,”com.android.vending.CHECK_LICENSE”,”android.permission.CALL_PHONE”>
	.....

4. Results and discussion. As depicted in the prediction phase, the risk of an unknown app is identified based on a request for permission analysis using rule-based classification. This section describes the results of the risk classification process. The JRip data mining model identified underlying patterns as “IF-THEN” rules displayed in Table 4.1. The importance of rules has been determined based on support and confidence measures. Comparative support is the frequency with which the antecedent of a rule appears in the training dataset. The outcome of the JRip model evaluation with 5-fold cross-validation showed a precision of 0.935, recall of 0.957, and f-measure of 0.946. These obtained rules discovered the substance of predictor as benign frequency, malware frequency, and malware ratio.

Glimpsing at the extracted rules, almost equal malware and benign frequency values directly correlate with low-peril events. Similarly, negligible malware frequency correlates with no peril, and negligible benign frequency correlates with high peril. The default rule specifies that no peril is induced when the predecessor of any other rule does not match.

Additional analysis has been performed for pattern mining using PrefixSpan to detect frequent sequences of permissions for malware and benign apps. A sample of the output results for malware & benign applications is given in Table 4.2. These frequent sequences showed that a relevant portion of the malware applications requested ”READ_PHONE_STATE”, ”ACCESS_NETWORK_STATE”, ”SEND_ SMS”, ”INTERNET”, ”WRITE_EXTERNAL_STORAGE”, ”INSTALL_PACKAGES”, and ”DE LETE_PACKAGES” permissions together. Similarly, a relevant portion of the benign applications requested ”READ_PHONE_STATE”, ”WRITE_EXTERN , ”ACCESS_NETWORK_STATE”, and ”INTERNET” permissions together. Notable malware applications requested only a single permission Internet or Internet and Phone State Access together to perform malicious

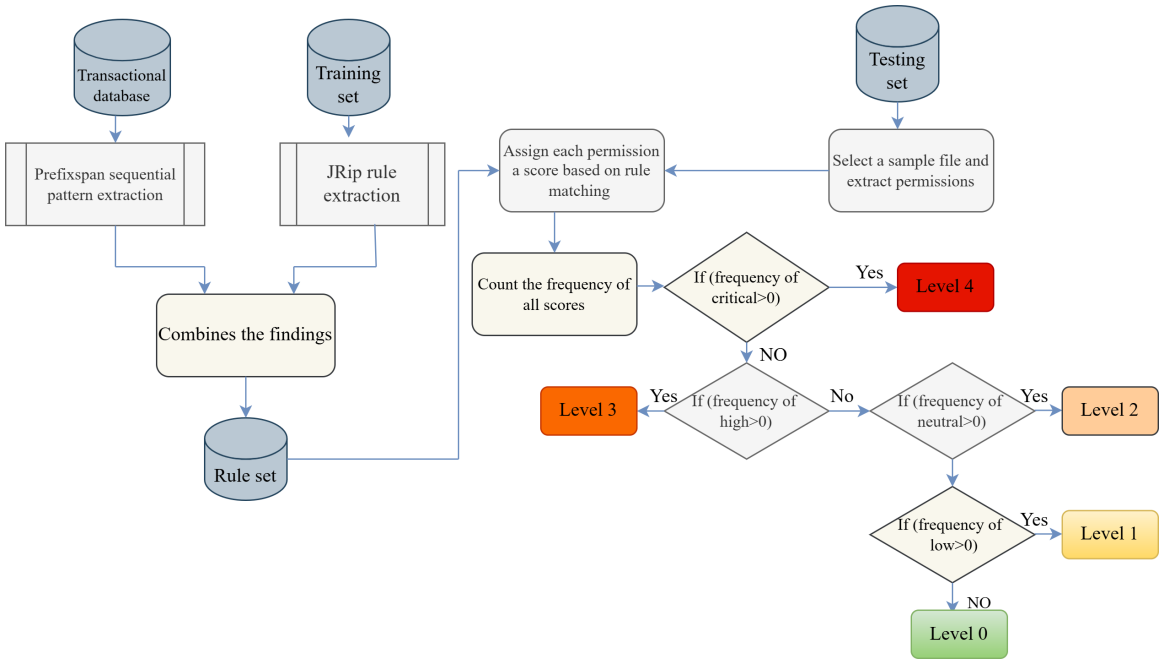

Fig. 3.6: Rule-based prediction

Table 4.1: Rules identified by JRip

Rule	**Comparative Support**
VL malware ratio and M benign frequency ⇒ medium peril	11
MD malware ratio ⇒ medium peril	18
[MD,M] benign frequency and [MD,M] malware frequency ⇒ low peril	20
N malware frequency ⇒ no peril	75
L malware ratio ⇒ medium peril	12
VL benign frequency and M malware frequency ⇒ very low peril	31
N benign frequency ⇒ high peril	33
M malware ratio ⇒ very low peril	22
[VL,L] benign frequency and [VL,L] malware frequency ⇒ low peril	29
default rule ⇒ no peril	45

activity. Further, malware applications frequently requested to read launcher settings and manage shortcut permissions.

The frequent benign sequences revealed that benign applications typically focused on providing basic functionality and enhancing the user experience. The frequent sequence of permissions ”RECEIVE_ BOOT_COM PLETED”, ”WAKE_LOCK”, ”VIBRATE”, and ”WRITE_EXTERNAL_STORAGE” together produce a responsive user experience. The combination of permissions ”CHANGE_WIFI_STATE”, ”BLUETOOTH”, and ”BLUETOOTH_ADMIN” enhances wireless connection functionality. Although some similarities are found in the permission requests of malware & benign apps, the frequent sequence & combination of permissions assist in revealing distinct patterns for both. Permission combinations for malware are typically broader and more sensitive, whereas benign app combinations are more closely aligned with user-focused functionalities.

Based on rule-based segmentation and permission analysis through frequent sequences, a rule set shown in Table 4.3 is constructed to identify the risk level of an app. The permissions related to a feature have been grouped together and assigned a risk score based on their uses in performing malicious and normal activity and part of frequent sequences of benign and malware apps.

Table 4.2: Sample of sequential rules extracted by PrefixSpan algorithm

Frequent sequences of permissions for malware apps	”android.permission.INTERNET”,”android.permission.READ_PHONE_STATE” support = 16
	”android.permission.READ_PHONE_STATE”,”android.permission.ACCESS_NETWORK_STATE”, ”android.permission.SEND_SMS”,”android.permission.INTERNET”,”android.permission.WRITE_E XTERNAL_STORAGE”,”android.permission.INSTALL_PACKAGES”,”android.permission.DELET E_PACKAGES” support = 80
	”com.android.launcher.permission.UNINSTALL_SHORTCUT”,”com.android.launcher.permission.RE AD_SETTINGS”,”com.htc.launcher.permission.READ_SETTINGS”,”com.motorola.launcher.permiss ion.READ_SETTINGS”,”com.motorola.dlauncher.permission.READ_SETTINGS” support = 23
	”android.permission.INTERNET” support = 16
	”android.permission.ACCESS_NETWORK_STATE”,”android.permission.READ_PHONE_STATE”, ”com.fede.launcher.permission.READ_SETTINGS”,”com.lge.launcher.permission.INSTALL_SHORT CUT”,”com.lge.launcher.permission.READ_SETTINGS”,”com.motorola.dlauncher.permission.INST ALL_SHORTCUT”,”com.motorola.launcher.permission.INSTALL_SHORTCUT” support = 15
	”android.permission.INTERNET”,”android.permission.ACCESS_NETWORK_STATE” support=20
	”com.fede.launcher.permission.READ_SETTINGS”,”com.lge.launcher.permission.READ_SETTINGS” ,”org.adw.launcher.permission.READ_SETTINGS”,”com.motorola.launcher.permission.INSTALL_SH ORTCUT”,”com.motorola.dlauncher.permission.INSTALL_SHORTCUT”,”com.lge.launcher.permissi on.INSTALL_SHORTCUT” support = 18
	”android.permission.ACCESS_WIFI_STATE”,”android.permission.READ_PHONE_STATE”sup.=15
	”android.permission.ACCESS_NETWORK_STATE”,”android.permission.ACCESS_WIFI_STATE”, ”android.permission.RECEIVE_BOOT_COMPLETED”,”android.permission.VIBRATE”,”android.p ermission.WAKE_LOCK” support = 18
	”android.permission.ACCESS_WIFI_STATE”,”android.permission.INTERNET”,”android.permission .READ_PHONE_STATE”,”android.permission.RECEIVE_BOOT_COMPLETED”,”android.permissi on.SEND_SMS”,”android.permission.WRITE_EXTERNAL_STORAGE” support = 17
	”android.permission.INSTALL_PACKAGES”,”android.permission.READ_PHONE_STATE”sup.=40
	”android.permission.READ_PHONE_STATE”,”android.permission.RECEIVE_BOOT_COMPLETE D”,”android.permission.SEND_SMS”,”android.permission.WAKE_LOCK”,”android.permission.WRI TE_EXTERNAL_STORAGE” support = 28
	”android.permission.ACCESS_WIFI_STATE”,”android.permission.READ_PHONE_STATE”,”com.an droid.launcher.permission.INSTALL_SHORTCUT”,”com.android.launcher.permission.UNINSTALL_ SHORTCUT”,”com.lge.launcher.permission.INSTALL_SHORTCUT”,”com.lge.launcher.permission.R EAD_SETTINGS” support = 15
	”android.permission.ACCESS_NETWORK_STATE”,”android.permission.ACCESS_WIFI_STATE”, ”android.permission.INTERNET”,”android.permission.READ_PHONE_STATE”,”android.permissio n.RECEIVE_BOOT_COMPLETED”,”android.permission.WAKE_LOCK”,”android.permission.WRI TE_EXTERNAL_STORAGE” support = 31
Frequent sequences of permissions for benign apps	”android.permission.INTERNET”,”android.permission.ACCESS_NETWORK_STATE”,”android.per mission.WRITE_EXTERNAL_STORAGE”,”android.permission.READ_PHONE_STATE”support=45
	”android.permission.RECEIVE_BOOT_COMPLETED”,”android.permission.WAKE_LOCK”,”andro id.permission.VIBRATE”,”android.permission.WRITE_EXTERNAL_STORAGE” support = 19
	”android.permission.ACCESS_COARSE_LOCATION”,”android.permission.ACCESS_FINE_LOCATI ON” support = 23
	”android.permission.READ_CONTACTS”,”android.permission.CALL_PHONE”,”android.permission .READ_SMS”,”android.permission.WRITE_SMS” support = 15
	”android.permission.ACCESS_WIFI_STATE”,”android.permission.READ_PHONE_STATE”,”androi d.permission.WRITE_EXTERNAL_STORAGE” support = 25
	”com.android.browser.permission.READ_HISTORY_BOOKMARKS”,”com.android.browser.permissi on.WRITE_HISTORY_BOOKMARKS” support = 10
	”android.permission.CHANGE_WIFI_STATE”,”android.permission.BLUETOOTH”,”android.permis sion.BLUETOOTH_ADMIN” support = 15
	”android.permission.WRITE_EXTERNAL_STORAGE”,”com.android.vending.CHECK_LICENSE” support = 23
	”android.permission.CHANGE_WIFI_STATE”,”android.permission.WRITE_SETTINGS”,”android.p ermission.WRITE_SYNC_SETTINGS” support = 17
	”android.permission.GET_TASKS”,”android.permission.RESTART_PACKAGES” support = 16
	”android.permission.READ_CONTACTS”,”android.permission.WRITE_CONTACTS”,”android.per mission.WRITE_EXTERNAL_STORAGE” support = 15
	”com.android.vending.BILLING”,”android.permission.WRITE_EXTERNAL_STORAGE”support=20
	”android.permission.GET_ACCOUNTS”,”android.permission.USE_CREDENTIALS”,”android.perm is- sion.MANAGE_ACCOUNTS” support = 18
	”android.permission.VIBRATE”,”com.android.launcher.permission.INSTALL_SHORTCUT”sup.=14

Table 4.3: Overview of the rule set used for evaluation

Rule statement	**Description**	**Assign score**
”android.permission.WRITE_SMS”/”android.permission.READ_SMS”/”android.permission.RECEIVE_SMS”/”android.permission.SEND_SMS”	Allows control of SMS communication such as message reading, sending new ones, and spoiling incoming messages.	High/ Neutral/Low
”android.permission.ACCESS_COARSE_UPDATES”/”android.permission.ACCESS_FINE_LOCATION”	Allows to access location information for respective scenarios.	Default
”com.motorola.dlauncher.permission.READ_SETTINGS”/”com.motorola.dlauncher.permission.INSTALL_SHORTCUT”/”com.motorola.launcher.permission.INSTALL_SHORTCUT”/”com.motorola.launcher.permission.READ_SETTINGS”/”com.lge.launcher.permission.INSTALL_SHORTCUT”/”com.lge.launcher.permission.READ_SETTINGS”	Allows the creation of shortcuts on the home screen of the respective launcher and the reading of the configuration of the respective launcher.	Critical
”android.permission.READ_CONTACTS”/”android.permission.WRITE_CONTA CTS”/”android.permission.CALL_PHONE”	Allows control of contact management and calling features.	Low
”android.permission.DELETE_PACKAGES”	Allows system-level access to delete other apps (packages) from the device.	Critical
”android.permission.ACCESS_NETWORK_STATE”/”android.permission.INTERNET”	Allows to check network availability and connect to the internet.	Default
”android.permission.CHANGE_WIFI_STATE”/”android.permission.ACCESS_WIFI_STATE”/”android.permission.CHANGE_NETWORK_STATE”	Allows to control the Wi-Fi and network status of the device.	Neutral/ Low
”android.permission.MODIFY_PHONE_STATE”/”android.permission.PROCESS_OUTGOING_CALLS”	Allows control of the phone’s state and behaviour, and outgoing calls.	Neutral
”android.permission.ACCESS_LOCATION_EXTRA_COMMANDS”/”android.permission.ACCESS_COARSE_LOCATION”/”android.permission.ACCESS_MOCK_LOCATION”	Allows the creation of mock location providers and access to location information and additional location provider information.	Neutral/ Low
”com.software.application.permission.C2D_MESSAGE”/”com.rvo.plpro.permission.C2D_MESSAGE”/”com.p1.chompsms.permission.C2D_MESSAGE”/”com.samsungmobileusa.magnacarta.permission.C2D_MESSAGE”	Allows access to the communication channels of respective apps, such as allowing access to push notifications and messages.	Critical
”android.permission.WRITE_SETTINGS”/”android.permission.WRITE_SECURE_SETTINGS”/”android.permission.CHANGE_CONFIGURATION”/”android.permission.MODIFY_AUDIO_SETTINGS”	Allows to modify the system settings for respective requests.	Low/ Default
”android.permission.WRITE_EXTERNAL_STORAGE”	Allows writing on device’s shared storage locations.	Default
”android.permission.STATUS_BAR”/”android.permission.INTERNAL_SYSTEM_WINDOW”/”android.permission.ADD_SYSTEM_SERVICE”	Allows modification of core system behaviours, overlaying of system UI elements, and addition of new system services.	Critical
”com.android.browser.permission.READ_HISTORY_BOOKMARKS”/”com.android.browser.permission.WRITE_HISTORY_BO OKMARKS”	Allows to read and modify the significant insight of online activities such as browsing history and stored bookmarks.	High/ Neutral
”android.permission.GET_TASKS”/”android.permission.KILL_BACKGROUND_PROCESSES”/”android.permission.RESTART_PACKAGES”	Allows to access the information of running tasks and control of other applications.	Low
”android.permission.REBOOT”/”android.permission.BACKUP”	Allows to control reboot and backup processes.	Critical
”android.permission.BROADCAST_SMS”/”android.permission.BROADCAST_WAP_PUSH”	Allows to control notifications of incoming SMS and WAP PUSH messages.	Low
”com.facebook.katana.provider.ACCESS”/”com.mominis.permission.preferences.provider.READ_WRITE”	Allows access to the respective app’s data providers.	Critical
”android.permission.UPDATE_DEVICE_STATS”/”android.permission.READ_PHONE_STATE”/”android.permission.DEVICE_POWER”	Allows to access device information such as phone status and power usage details, and to control statistics updates.	Low/ Default
”android.permission.FLASHLIGHT”/”android.permission.VIBRATE”/”android.permission.EXPAND_STATUS_BAR”/”android.permission.SET_WALLPAPER”/”android.permission.WAKE_LOCK”/”android.permission.DISABLE_KEYGUARD”	Allows to control the device features for respective requests.	Neutral/ Low/ Default
”android.permission.ACCESS_GPS”/”android.permission.ACCESS_LOCATION”	Allows to access device location information.	High

Continued on next page

Continued from previous page

Rule statement	Description	Assign score
”com.android.launcher.permission.READ_SETTINGS”/”com.android.launcher.permission.INSTALL_SHORTCUT”/”com.android.launcher.permission.UNINSTALL_SHORTCUT”/”com.fede.launcher.permission.READ_SETTINGS”/”org.adw.launcher.permission.READ_SETTINGS”/”com.htc.launcher.permission.READ_SETTINGS”	Allows to read information about the home screen setup, and to create and remove shortcuts on the home screen of the respective launchers.	Critical/ High/ Low
”android.permission.SET_DEBUG_APP”	Allows to configure another app for debugging.	Default
”android.permission.BROADCAST_STICKY”/”android.permission.READ_LOGS”	Allows to access device logs and broadcast persistent messages.	Neutral/ Low
”android.permission.WRITE_APN_SETTINGS”	Allows modification of APN (Access Point Name) network configuration settings.	Neutral
”android.permission.PROCESS_INCOMING_CALLS”/”android.permission.PROCESS_CALL”	Allows access to interact with and manage phone calls.	Critical
”android.permission.CAMERA”/”android.permission.GET_ACCOUNTS”/”android.permission.RECEIVE_BOOT_COMPLETED”/”android.permission.BLUETOOTH”	Allows to access different features and capabilities of the device.	Low/ Default
”android.permission.INSTALL_PACKAGES”	Allows to initiate the installation of other applications.	Critical
”android.permission.SYSTEM_ALERT_WINDOW”/”android.permission.RECORD_AUDIO”	Allows the creation of overlay windows and recording of conversations.	Low
”android.permission.BATTERY_STATS”/”android.permission.READ_OWNER_DATA”	Allows monitoring of device-specific information and power usage.	Critical
”com.android.vending.CHECK_LICENSE”/”com.android.vending.BILLING”	Allows to access services of Google Play Store.	Default
”android.permission.REORDER_TASKS”/”android.permission.SET_PROCESS_LIMIT”/”android.permission.SET_ALWAYS_FINISH”/”android.permission.CLEAR_APP_USER_DATA”/”android.permission.CLEAR_APP_CACHE”	Allows to control the application management.	Critical/ High
”android.permission.READ_EXTERNAL_STORAGE”/”android.permission.MOUNT_UNMOUNT_FILESYSTEMS”/”android.permission.PERSISTENT_ACTIVITY”	Allows control of the device’s storage.	Low
”android.permission.RECEIVE_WAP_PUSH”/”android.permission.WRITE_SECURE”/”android.permission.DELETE_CACHE_FILES”	Allows modification of secure system settings, deletion of cache files, and processing of WAP(Wireless Application Protocol) push messages.	Critical
”android.permission.RECEIVE_MMS”/”com.google.android.c2dm.permission.RECEIVE”	Allows to receive notifications from google servers and multimedia messages services.	Low

The main contribution of this research work is to classify Android apps into five different risk levels based on static behaviour analysis. An investigation is performed with benign AndroZoo apps and malware Drebin apps. To analyze the results, risk level 4 is classified as malware, and risk level 0 is classified as benign. First, all permissions whose benign frequency was zero with a positive malware frequency and whose benign frequency was less than or equal to one with a frequency of malware greater than or equal to 23 were placed at the critical level. With each decreasing risk level, permissions were added by increasing the benign frequency by 7 and reducing the malware frequency by 7. The results obtained from this analysis is shown in Table 4.4.

Then, three less-risky permissions were moved from critical to high level, and three high-risky permissions were moved from default to low level, the results of which are shown in Table 4.5. Once again, three less-risky permissions were moved from critical to high level, and three high-risky permissions were moved from default to low level, the results of which are shown in Table 4.6. Tables 4.4-4.6 show the matrix, where each

Table 4.4: Malware & benign risk level

Class	Level0	Level1	Level2	Level3	Level4	Total
Malware	86	123	524	57	560	1350
Benign	518	490	228	33	81	1350
Total	604	613	752	90	641	2700

Table 4.5: Malware & benign risk level

Class	Level0	Level1	Level2	Level3	Level4	Total
Malware	34	175	524	81	536	1350
Benign	355	653	228	51	63	1350
Total	389	828	752	132	599	2700

Table 4.6: Malware & benign risk level

Class	Level0	Level1	Level2	Level3	Level4	Total
Malware	24	185	524	101	516	1350
Benign	316	692	228	86	28	1350
Total	340	877	752	187	544	2700

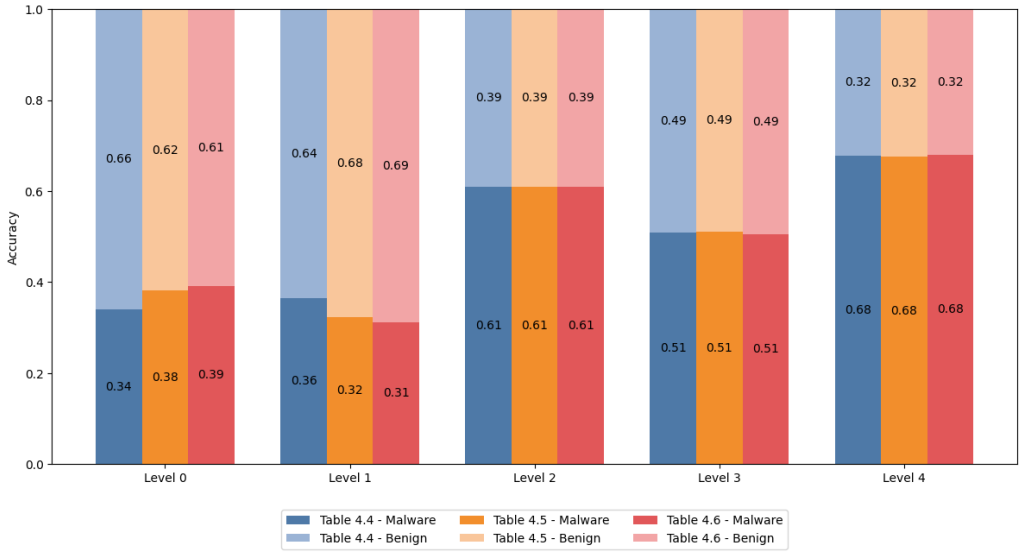

Fig. 4.1: Accuracy by different level

Table 4.7: Confusion matrix

Class	Malware	Benign	Total	Accuracy
Malware	true negative	false negative	1350	98.22 %
Benign	false positive	true positive	1350	97.92 %
Total	1354	1346	2700	98.07 %

cell represents the proposition of each class (Malware and Benign) with corresponding levels. Most malware samples have been predicted at higher risk levels, with a small portion misclassified at lower levels. Similarly, most benign samples have been correctly classified at lower risk levels, with a small portion misclassified at higher levels. The respective accuracy, recall, and precision for different risk levels are shown in Figs. 4.1-4.3.

The work performance has been evaluated with parameters sensitivity, specificity, and accuracy, as in Eqs. 4.1 to 4.3 & the confusion matrix shown in Table 4.7. True positive is the correct classification of a positive outcome, and false negative is the misclassification of a positive outcome as negative. Similarly, true negative is the correct classification of a negative outcome, and false positive is the misclassification of a negative outcome

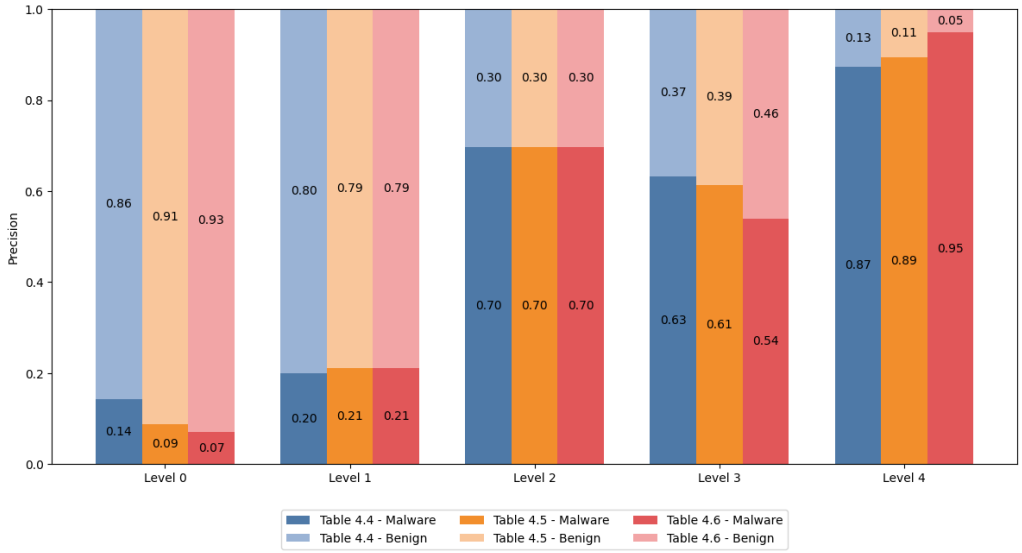

Fig. 4.2: Precision by different level

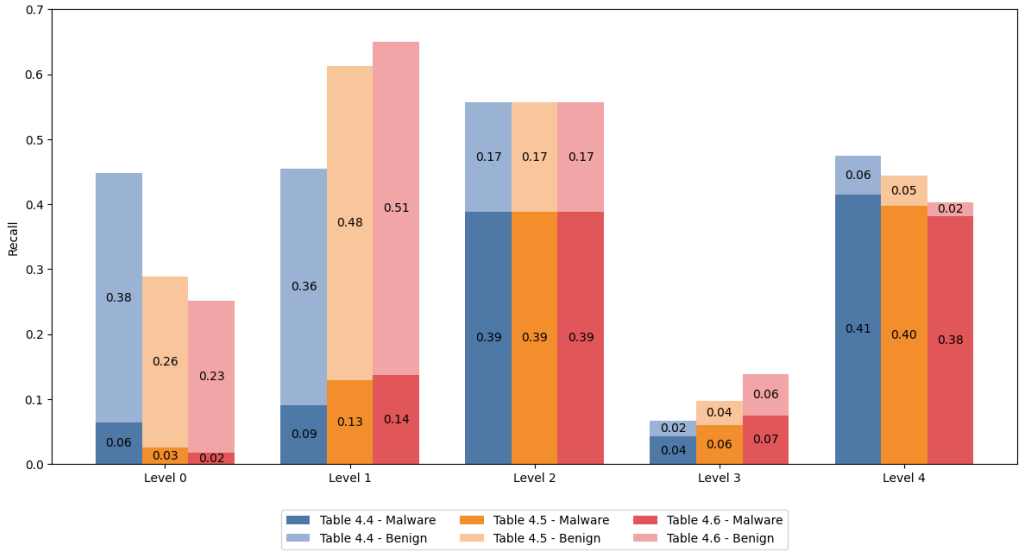

Fig. 4.3: Recall by different level

as positive.

$$Sensitivity = \frac{True\ positive}{Total\ positive} = 97.92\% \tag{4.1}$$

$$Specificity = \frac{True\ negative}{Total\ negative} = 98.22\% \tag{4.2}$$

Table 4.8: Statistical comparison of benign and malware apps at different risk levels

Malware	Mean: 270.0, Median: 185.0, Std: 210.39676803601333
Benign	Mean: 270.0, Median: 228.0, Std: 234.25797745220973
Pearson Correlation	-0.35015232234884325
Risk Level 0	Difference = -292, Ratio = 0.0759493670886076
Risk Level 1	Difference = -507, Ratio = 0.26734104046242774
Risk Level 2	Difference = 296, Ratio = 2.2982456140350878
Risk Level 3	Difference = 15, Ratio = 1.1744186046511629
Risk Level 4	Difference = 488, Ratio = 18.428571428571427
Chi-square statistic	1099.355365399867
p-value	1.04451108717828 64e-236

Table 4.9: Post-hoc Pairwise comparison

Level 0 vs Level 1	Chi2 Statistic: 32.95461756214981 p-value: 9.433542164126035e-09 Adjusted p-value (Bonferroni): 9.433542164126036e-08 Significant? Yes
Level 0 vs Level 2	Chi2 Statistic: 364.77674116066646 p-value: 2.5673262468768854e-81 Adjusted p-value (Bonferroni): 2.5673262468768853e-80 Significant? Yes
Level 0 vs Level 3	Chi2 Statistic: 144.4108830167345 p-value: 2.8890715869634067e-33 Adjusted p-value (Bonferroni): 2.8890715869634066e-32 Significant? Yes
Level 0 vs Level 4	Chi2 Statistic: 674.747436073428 p-value: 9.27222763054354e-149 Adjusted p-value (Bonferroni): 9.27222763054354e-148 Significant? Yes
Level 1 vs Level 2	Chi2 Statistic: 386.8297061006291 p-value: 4.055656994876432e-86 Adjusted p-value (Bonferroni): 4.0556569948764324e-85 Significant? Yes
Level 1 vs Level 3	Chi2 Statistic: 83.30100038987347 p-value: 7.046393820082156e-20 Adjusted p-value (Bonferroni): 7.046393820082156e-19 Significant? Yes
Level 1 vs Level 4	Chi2 Statistic: 727.7937629862088 p-value: 2.7035449277334716e-160 Adjusted p-value (Bonferroni): 2.7035449277334714e-159 Significant? Yes
Level 2 vs Level 3	Chi2 Statistic: 15.825440927996633 p-value: 6.94622342675924e-05 Adjusted p-value (Bonferroni): 6.94622342675924e-04 Significant? Yes
Level 2 vs Level 4	Chi2 Statistic: 124.59595077718842 p-value: 6.238825620876588e-29 Adjusted p-value (Bonferroni): 6.238825620876588e-28 Significant? Yes
Level 3 vs Level 4	Chi2 Statistic: 173.2655691717622 p-value: 1.4320964914080645e-39 Adjusted p-value (Bonferroni): 1.4320964914080645e-38 Significant? Yes

$$Accuracy = \frac{(True\ positive + True\ negative)}{(Total\ positive + Total\ negative)} = 98.07\% \tag{4.3}$$

The chi-square test has been conducted to analyze the relationship between risk levels and app type (benign/malware)—the statistical analysis observations for malware and benign apps are shown in Table 4.8. A negative Pearson Correlation value indicates the inverse relationship between benign and malware apps. The large chi-square statistics value (1099.36) with a vastly smaller p-value indicates strong evidence of a relationship between app type and risk level. Initial chi-square showed apps are not distributed randomly; there is a significant difference between malware and benign app distribution across risk levels. Further, a Post-hoc Pairwise comparison has been performed to know the significant differences between every possible pair of risk levels, the observations of which are shown in Table 4.9. These observations indicate significant differences between risk levels particularly at extreme levels.

Comparative analysis sheds light on the exploration process and the subsequent interpretation of the outcomes. Table 4.10 compares this work with the existing approaches based on the analytical framework, identification of risk categories, and data representatives. The results show that the presented work identified five risk levels for Android apps and performed well compared to existing works.

This research enhanced Android users' awareness of the need to understand the permission requests of apps, as allowing inappropriate permissions can put users at risk of malware attacks. The limitation is that this work focuses only on the permission requests of apps; if an app does not request any permissions, it becomes difficult to identify the exact nature of that app. Additional static features extracted from the manifest file analysis,

Table 4.10: Comparison with previous works

Approach	Analytical framework	Identified risk categories	Data Set
Probabilistic risk detector [11]	Static permissions analysis and artificial neural network model	Four (no, low, medium, and high risk)	3547
AndroShield [27]	Hybrid of static (code scanning) and dynamic (run-time behaviour) analysis for vulnerability detection	Three (low, medium, high)	70
RNPDroid [12]	Static permissions analysis and ANOVA and T-test	Four (no, low, medium, & high risk)	400
Focused on repacked malware samples [28]	Static source code analysis and fuzzy hash of reverse-engineered code	Three (benign, suspicious, malicious)	3490
PRAZdroid (proposed)	Static permissions analysis, demeanor rule mining (JRip), and frequent pattern identification (PrefixSpan)	Five (level 0, level 1, level 2, level 3, & level 4)	3100

such as intent filters, can be selected to extend this research.

5. Conclusion and future work. Android accessibility features and user-friendly nature make it an incredible platform for everyone. Although the Android ecosystem is growing and offers users a wide range of applications to cover every plausible aspect of life, there is a potential threat in the form of malware. Attackers influence Android users by injecting different menaces into Android applications. Malicious things pose as seemingly benign applications and cause disturbance by stealing data, causing system disruptions, and risking users' privacy. Promptly identifying and diminishing these risks remains a major challenge. It is complicated to determine the intentions of an app without using it, but every app requires permission authentication to access the user's device. Users are invited to grant an app's privileges through the requested permissions. Attackers mislead users to carry out malicious activities as other infiltration methods are nearly closed. The users ignore security concerns and allow these permissions because technical skills about the permissions and their impacts are needed to make a correct decision, and malicious apps may request permissions similar to benign ones. Therefore, this work analyzed the permissions requested by Android apps and identified risk levels. These risk levels are identified based on 210 extracted permissions, of which 33 permissions are requested only by malicious apps, 112 permissions are requested only by benign apps, and 65 permissions are requested by both malware & benign apps. In the future, the scope of analysis can be expanded to include monitoring resource usage, runtime behaviour of apps, intent activities, and network traffic patterns. In addition, big data analytics techniques can handle and process large amounts of app data efficiently.

REFERENCES

[1] Statista, *Google Play Store: number of apps 2023*, Accessed: Jul. 24, 2024. [Online]. Available: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store

[2] Y. Ishii, T. Watanabe, F. Kanei, Y. Takata, E. Shioji, M. Akiyama, T. Yagi, B. Sun, and T. Mori, *Understanding the security management of global third-party Android marketplaces*, in Proceedings of the 2nd ACM SIGSOFT International Workshop on App Market Analytics, Sep. 2017, pp. 12–18. doi: 10.1145/3121264.3121267.

[3] PurpleSec, *2023 Cyber Security Statistics: The Ultimate List Of Stats, Data and Trends*, Accessed: Jan. 06, 2024. [Online]. Available: https://purplesec.us/resources/cyber-security-statistics/.

[4] G. Shrivastava, P. Kumar, D. Gupta, and J. J. P. C. Rodrigues, *Privacy issues of android application permissions: A literature review*, Trans. Emerg. Telecommun. Technol., vol. 31, no. 12, p. e3773, Dec. 2020, doi: 10.1002/ett.3773.

[5] A. Dahiya, S. Singh, and G. Shrivastava, *Malware Detection Insights, Mechanisms and Future Perspectives for Android Applications*, in Innovative Computing and Communications, vol. 1021, Singapore: Springer Nature Singapore, 2024, pp. 381–403. doi: 10.1007/978-981-97-3591-4_31.

[6] M. Damshenas, A. Dehghantanha, K.-K. R. Choo, and R. Mahmud, *M0Droid: An Android Behavioral-Based Malware Detection Model*, J. Inf. Priv. Secur., vol. 11, no. 3, pp. 141–157, Jul. 2015, doi: 10.1080/15536548.2015.1073510.

[7] L. Bao, D. Lo, X. Xia, and S. Li, *Automated Android application permission recommendation*, Sci. China Inf. Sci., vol. 60, no. 9, p. 092110, Sep. 2017, doi: 10.1007/s11432-016-9072-3.

[8] Md. Y. Karim, H. Kagdi, and M. Di Penta, *Mining Android Apps to Recommend Permissions*, in 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Suita: IEEE, Mar. 2016, pp. 427–437. doi: 10.1109/SANER.2016.74.

[9] J. Xiao, S. Chen, Q. He, Z. Feng, and X. Xue, *An Android application risk evaluation framework based on minimum permission set identification*, J. Syst. Softw., vol. 163, p. 110533, May 2020, doi: 10.1016/j.jss.2020.110533.
[10] M. Deypir and A. Horri, *Instance based security risk value estimation for Android applications*, J. Inf. Secur. Appl., vol. 40, pp. 20–30, Jun. 2018, doi: 10.1016/j.jisa.2018.02.002.
[11] M. Dhalaria and E. Gandotra, *Risk Detection of Android Applications Using Static Permissions*, in Advances in Data Computing, Communication and Security, Singapore: Springer Nature, 2022, pp. 591–600. doi: 10.1007/978-981-16-8403-6_54.
[12] K. Sharma and B. B. Gupta, *Mitigation and risk factor analysis of android applications*, Comput. Electr. Eng., vol. 71, pp. 416–430, Oct. 2018, doi: 10.1016/j.compeleceng.2018.08.003.
[13] H. AlOmari, Q. M. Yaseen, and M. A. Al-Betar, *A Comparative Analysis of Machine Learning Algorithms for Android Malware Detection*, Procedia Comput. Sci., vol. 220, pp. 763–768, Jan. 2023, doi: 10.1016/j.procs.2023.03.101.
[14] Y. Sharma and A. Arora, *IPAnalyzer: A novel Android malware detection system using ranked Intents and Permissions*, Multimed. Tools Appl., Mar. 2024, doi: 10.1007/s11042-024-18511-6.
[15] M. Upadhayay, A. Sharma, G. Garg, and A. Arora, *RPNDroid: Android Malware Detection using Ranked Permissions and Network Traffic*, in 2021 Fifth World Conference on Smart Trends in Systems Security and Sustainability (WorldS4), London, United Kingdom: IEEE, Jul. 2021, pp. 19–24. doi: 10.1109/WorldS451998.2021.9513992.
[16] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, *MADAM: Effective and Efficient Behavior-based Android Malware Detection and Prevention*, IEEE Trans. Dependable Secure Comput., vol. 15, no. 1, pp. 83–97, Jan. 2018, doi: 10.1109/TDSC.2016.2536605.
[17] D. Naga Malleswari, A. Dhavalya, V. Divya Sai, and K. Srikanth, *A detailed study on risk assessment of mobile app permissions*, Int. J. Eng. Technol., vol. 7, no. 1.1, p. 297, Dec. 2017, doi: 10.14419/ijet.v7i1.1.9706.
[18] A. Dahiya, S. Singh, and G. Shrivastava, *Android malware analysis and detection: A systematic review*, Expert Syst., p. e13488, Oct. 2023, doi: 10.1111/exsy.13488.
[19] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, *Drebin: Effective and Explainable Detection of Android Malware in Your Pocket*, in Proceedings 2014 Network and Distributed System Security Symposium, San Diego, CA: Internet Society, 2014. doi: 10.14722/ndss.2014.23247.
[20] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, *AndroZoo: collecting millions of Android apps for the research community*, in Proceedings of the 13th International Conference on Mining Software Repositories, Austin Texas: ACM, May 2016, pp. 468–471. doi: 10.1145/2901739.2903508.
[21] K. Sharma and B. B. Gupta, *Towards Privacy Risk Analysis in Android Applications Using Machine Learning Approaches*, Int. J. E-Serv. Mob. Appl. IJESMA, vol. 11, no. 2, pp. 1–21, Apr. 2019, doi: 10.4018/IJESMA.2019040101.
[22] M. Khari, R. Dalal, U. Misra, and A. Kumar, *AndroSet: An automated tool to create datasets for android malware detection and functioning with WoT*, in Smart Innovation of Web of Things, CRC Press, 2020, pp. 187–206. doi: 10.1201/9780429298462-11.
[23] R. Wiśniewski and C. Tumbleson, *A Tool for Reverse Engineering Android Apk Files*, (2017). Available online: https://ibotpeaches.github.io/apktool/.
[24] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, *The WEKA data mining software: an update*, SIGKDD Explor Newsl, vol. 11, no. 1, pp. 10–18, Nov. 2009, doi: 10.1145/1656274.1656278.
[25] W. W. Cohen, *Fast Effective Rule Induction*, in Machine Learning Proceedings 1995, Elsevier, 1995, pp. 115–123. doi: 10.1016/B978-1-55860-377-6.50023-2.
[26] J. Pei, J. Han, B. Mortazavi-Asl , J. Wang, H. pinto, Q. Chen, U. Dayal and M.-C. Hsu, *Mining sequential patterns by pattern-growth: the PrefixSpan approach*, IEEE Trans. Knowl. Data Eng., vol. 16, no. 11, pp. 1424–1440, Nov. 2004, doi: 10.1109/TKDE.2004.77.
[27] A. Amin, A. Eldessouki, M. T. Magdy, N. Abdeen, H. Hindy, and I. Hegazy, *AndroShield: Automated Android Applications Vulnerability Detection, a Hybrid Static and Dynamic Analysis Approach*, Information, vol. 10, no. 10, p. 326, Oct. 2019, doi: 10.3390/info10100326.
[28] H. Ali, K. Batool, M. Yousaf, M. I. Satti, S. Naseer, S. Zahid, A. A. Gardezi, M. Shafiq, and J. -G. Choi, *Security Hardened and Privacy Preserved Android Malware Detection Using Fuzzy Hash of Reverse Engineered Source Code*, Secur. Commun. Netw., vol. 2022, pp. 1–11, Sep. 2022, doi: 10.1155/2022/7972230.

Edited by: Kavita Sharma
Special issue on: Recent Advance Secure Solutions for Network in Scalable Computing
Received: Jul 29, 2024
Accepted: Nov 26, 2024