Journal, Advances in Mathematical & Computational Sciences
Vol. 12 No. 2, 2024 Series
www.isteams.net/mathematics-computationaljournal

**Journal of Advances in Mathematical & Computational Sciences**
An International Pan-African Multidisciplinary Journal of the SMART Research Group
International Centre for IT & Development (ICITD) USA
© Creative Research Publishers
Available online at https://www.isteams.net/ mathematics-computationaljournal.info
CrossREF Member Listing - https://www.crossref.org/06members/50go-live.html

# Software-Based Risk Level Analysis of Uses Permission On Android Platform

[1 2]*Edward Udo & Olivia Nwose
[1]Department of Computer Science, University of Uyo, Uyo, Nigeria
[2]TETFund Centre of Excellence in Computational Intelligence Research
University of Uyo, Uyo, Nigeria
E-mail: [1]edwardudo@uniuyo.edu.ng
Phone: +2348023339501

## ABSTRACT

The Android platform's open nature grants significant flexibility to application developers but also introduces privacy and security challenges. Users are frequently prompted to grant various permissions to applications, often without a clear understanding of the associated risks. Current application permission models may not adequately communicate the potential risks to users. Consequently, users may inadvertently grant excessive permissions, exposing their personal data and devices to potential threats. This work addresses the need for a comprehensive risk-level analysis of uses permissions within the Android platform and communicates potential security and privacy concerns to users. This is done by developing the RiLUP application, using Kotlin programming language and JetPack Compose, for the analysis of the risk level of uses permission on Android platform. Android classes, functions, and methods were used to achieve the set objectives. The method GET_META_DATA of the class get Installed Applications nested in the Package Manager class was used to get the list of applications on a device. Another nested class of the Package Manager class, get Package Info, and the method GET_PERMISSION were used to retrieve the permissions from each application. A risk algorithm based on points allocation system was developed to give a risk score taking cognizance of the number of sensitive permissions required and the number granted. The risk score and risk level were displayed on the users screen, using the colours red, blue and green to represent high-risk, medium-risk and safe applications respectively. RiLUP was evaluated by analyzing 40 applications from Google Play Store with a risk score showing 27 applications as safe, , 9 as medium-risk and 4 as high-risk. A domain name “olivianwose.dev” was bought from Name Cheap to host a site for easy sharing of the RiLUP APK file.

**Keywords:** Risk Level, Uses Permission, Android, Mobile Applications, Smartphones, Kotlin, JetPack Compose

## 1. INTRODUCTION

The openness of the Android system and the popularity of Android mobile devices have increased the number of applications (apps) developed in the last ten years. This leads to increase in personal data leakage (Guaman et al., 2021). These mobile devices have increasingly tangled with almost all aspect of human lives since they provide functionalities that help them to carry out their day-to-day activities (Yilmaz and Davis, 2023). These provided functionalities invariably encourage the increase in the number of users of Android platforms in recent years (Wang et al., 20213). As at April, 2024, Google play, the largest Android application store, has more than 2.3 million android applications (AppBrain, 2024). The increase in the number of users and Android apps has attracted the attention of malware developers. When these mobile devices are been used, personal data such as location, photographs, contacts etc, are being transferred from one mobile device to another, which increases the risk of personal information leakages (Kim et al., 2023) and mobile attacks (Ashawa and Morris, 2021).

Android, the most widely used mobile operating system with the highest number of users worldwide (Ashawa and Morris, 2021), is an open development platform. Open, not in the sense that everyone can contribute while a version is being developed, but in the sense that Android becomes open when its final source code is made available to the general public. This implies that after it has been made open, anyone can modify the code. Thus, Android's open and adaptable platform is more vulnerable to cyber attacks. The core building blocks, or fundamental Android components, are activities, views, intents, services, content providers, fragments, and AndroidManifest.xml (Wu and Liu, 2019).

An Android app is a software programme created to operate on an Android device or emulator. Developers may download the Android software development kit (SDK) from the Android website. SDK includes tools, sample code, and relevant documents for creating Android apps. Android Package or APK (also known as Android Application Package or Android Package Kit), is a file format used by Android to share and install applications. As a result, an APK has every component an application requires to successfully install on an Android device. Information about applications (also called application code), user interfaces (UIs), classes, resources, services, metadata, layout-related information and the manifest file are all compressed in the APK file, which is a zip file.

The security system for Android was created with various layers to provide flexibility, particularly for developers to create safe applications, and adequate protection for all platform users. Android offers some security features, such as the application sandbox and permission model, to safeguard users. Each application runs in an isolated sandbox (Alshehri, 2019), which is a managed and constrained environment to run and execute code. With the aid of this environment, developers may isolate and safeguard system resources from malware and other forms of online danger. Android makes use of security framework that allows apps to make a request for permissions to access sensitive end-users data (contacts, messages, call logs and other device's features like camera, Wi-Fi, location etc) (Xiao et al., 2020). The permission-based approach used by Android is intended to prevent the spread of malware while allowing end-users to grant permission to any app that wants to use their sensitive data. Granting these permissions rely on the users' ability to understand the application's functionality. However, in most cases, users are unaware that some applications may give extra permissions to unrelated parties (Alenezi and Almomani, 2017).

The granted permissions are installed on the users' mobile devices and can run on the mobile device's background collecting and transmitting data between applications which have the same permissions privileges and fall under the same permission groups, without the users' knowledge (Yilmaz and Davis, 2023). Developers of mobile platforms have implemented permissions models to evade these background processes, but apps designers, developers, providers and other third parties are not adhering to the general data protection policies, thus can access sensitive data on mobile platforms without the consent of the end-users ((Yilmaz and Davis, 2023). The security of Android is dependent on the effectiveness of the permission-based mechanism it uses (Xiao et al., 2020). With applications built for modern versions of Android, one need not grant any permission upon installing an application. Instead, an application will ask for permissions as it needs them (Stegner, 2023). According to Developer (2023), in the current Android version 13, the system permissions are split into three protection levels which are specified in the manifest file:

- Normal Permission: This permission allows access to information and activities outside the sandbox of an application, but poses a very minimal threat to user privacy or the functionality of other apps. It does not immediately threaten user's privacy. Normal permissions are automatically granted by the system. Example, INTERNET, SET_ALARM, SET_WALLPAPER, and WAKE_LOCK.
- Signature Permission: The device will automatically authorize signature permission if the application requesting it is signed with the same certificate as the application that declared the permission. Example WRITE_SETTINGS, WRITE_VOICEMAIL
- Runtime Permission: This permission is usually referred to as dangerous permission because it offers applications more access to restricted data or permits it to carry out prohibited actions that have a greater impact on the system and other applications. Example READ_CONTACTS, READ_SMS, RECEIVE_SMS, ACCESS_FINE_LOCATION, and ACCESS_COARSE_LOCATION

For a long time now, Android has been a major target of malicious apps because of its vulnerable permission mechanism (Huang et al., 2015; Zhang et al., 2016). There are enormous risks with regard to these permissions because up to 97% of malicious mobile malware targets Android platforms (Olukoya, 2019). The Android permission model does not hinder privilege abuse or information leakage. In other words, the permission model is not well designed to provide sufficient means of control over an application's activities and specify what private information or resources are accessible to the application. Thus, many attacks could possibly occur by exploiting critical permissions, such as privilege escalation sabotages (Alshehri, 2019). Google added the granting of risky permissions at runtime and displayed the different permission categories in Android 6.0 and later published versions. Google does not, however, provide thorough explanations of how they determined whether certain permissions were safe or risky.

The list of permission groups, the related permissions and their permission levels are shown in Table 1.

Table 1: Permission Groups, Related Permissions and their Levels (Source: Alazab et al., 2020)

| Permission Groups | Related Permissions | Permission Level |
|---|---|---|
| Alarm | Set_Alarm, Set_Time_Zone | Normal permission |
| Microphone | Record_Audio | Dangerous Permission |
| Bluetooth | Bluetooth, Bluetooth_Admin | Normal permission |
| Calendar | Read_Calendar, Write_Calendar | Dangerous Permission |
| Camera | Camera | Dangerous Permission |
| Contacts | Get_Accounts,Read_Contacts, Write_Contacts | Dangerous Permission |
| Location | Access_Coarse_Location, Access_Fine_Location | Dangerous Permission |
| Network | Access_Network_State, Access_ Notification_Policy, Access_Wi-Fi_State, Change_Wi-Fi_Multicast_State, Change_Wi-Fi_State | Normal Permission |
| Phone | Read_Phone_State, Read_Phone_Numbers, Call_Phone, Answer_Phone_Calls, Read_Call_Log, Write_Call_Log, Add_Voicemail, Process_Outgoing_Calls | Dangerous Permission |
| Sensors | Body_Sensors | Dangerous permission |
| SMS | Receive_SMS, Read_SMS, Receive_Wap_Push, and Receive_MMS | Dangerous permission |
| Storage | Write_External_Storage, Read_External_Storage | Dangerous permission |

This work therefore seeks to carry out a thorough, organized examination and analysis of the permissions system for the Android platform by developing a dedicated application called RiLUP (Risk Level Analysis of Uses Permissions) to analyze the already installed apps on the Android devices for risk related to the permissions they request, provide risk level and appropriate score based on permissions requested by each installed apps.

## 2. LITERATURE REVIEW

Yilmaz and Davis (2023) looked at permissions that are inbuilt in the Android mobile devices and showed how they can expose sensitive user information; identify the behavior of the user and how these information are shared among other applications. They did this by statistically analyzing the permissions of 10 first party Android applications and investigated their actual purpose and privacy risk. The permissions were categorized into 5 asset groups with risk levels of low to high which were used to form a risk model. Xiao et al., (2020) proposed minimum permission for Android (MPDroid), which combined static analysis and collaborative filtering approaches to identify the minimum permissions for an Android app based on the permissions requested by the app and API-used code permission. Given any app, MPDroid uses first the collaborative filtering approach to see the permissions granted for by the app, then uses the static analysis approach to identify the set of permissions that app actually needs. These approaches utilized Latent Dirichlet Allocation (LDA) to categorize applications into different topics based on their descriptions and API usage. MPDroid employed apktool to extract APK files, read permissions from Android Manifest.xml, and utilized Androguard to statistically analyz the permissions of the applications' code. MPDroid finally evaluated the over-privileged risk by looking at the extra permissions granted by the app. They conducted experiments on 16,343 well-known apps (benign and malicious) gotten from Google Play Store. The results showed that MPDroid performed significantly well.

Alazab et al., (2020) proposed a system for recognizing modified applications by analyzing the frequency distributions of API calls and permissions between two applications. The system employed an automated process using scoring and grouping techniques to identify crucial API calls in Android malware. The mobile malware analysis comprised three phases: pre-processing, extraction and a grouping phase that highlighted nuances in malicious API call usage. API calls were categorized into groups like ambiguous, dangerous, and disruptive, focusing on those prevalent in malicious apps and deeming those in benign apps irrelevant. To ensure consistent classifier weights, term frequency normalization was implemented. Classifiers performance was evaluated using standard metrics like accuracy, precision, recall, and F-measure. Five machine learning techniques were applied through 10-fold cross-validation, including random forest, J48, random tree, k-nearest neighbors, and naive Bayes. The results showed the proposed strategy successfully identified mobile malware, achieving an F-measure of 94.3%.

Alshehri et al., (2019) proposed Permission Usage and Risk Estimation for Android (PUREDroid), which assessed the security risk posed by the permissions of Android applications and the amount of harm caused by approving unnecessary permission requests. AndroZoo, an existing malware dataset for Android, served as the basis for the evaluation of PUREDroid. Using apktool, each Android application package (APK) was assessed, and permissions within the application were extracted. After reviewing the permissions, the risk function assigned a score to each of the permission depending on how frequently it was requested by both safe and malicious applications. One of three regions—a low-risk region, a moderate-risk region, or a high-risk region was assigned to the risk score.

Dini et al., (2018) introduced the Multi-Criteria Application Evaluator of Trust for Android (MAETROID), a framework designed to calculate the credibility of Android applications before installation. MAETROID deployed the Analytic Hierarchy Process (AHP) to merge criteria including metadata, requested permissions, and statistical assessments. The established criteria were global threat score based on declared rights, marketplace, developer reputation, user rating and number of downloads. The global threat score was derived from parsing the application's manifest file, considered declared permissions, resource access, and potential operations. After calculating criteria values, AHP was implemented. MAETROID's effectiveness was tested on a dataset of 9,804 Google Play applications and 1,242 known malware programs from the Genome database. Notably, 77.37% of Google Play applications posed no security threats, while 22% exhibited concerns, often due to low download numbers; genome applications did not receive a trusted rating.

Shen et al., (2018) proposed a framework comprising malicious application detection and permission recommendations for Android. The malicious application detection component evaluated installation risks by using a Random Forest-based classifier trained on permissions and intent filters extracted from AndroidManifest.xml. Androgurad, an open-source tool, facilitated static analysis, and Python Beautiful Soup library was used to extract intent-filter features like permissions and broadcast receivers. Feature vectors, derived from extracted features, were utilized to categorize applications and calculate permission risk scores. Risk scores were computed using a dataset of 6,400 benign applications. In an experiment with 10,979 good applications and 3,205 bad applications, the proposed approach demonstrated a true-positive rate (TPR) of 86.8% and a false-positive rate (FPR) of 0.99%.

Previous research and resources were based on certain versions of the Android operating system, which makes them almost obsolescence when new Android versions are released. Google often upgrades the Android operating system, and these modifications directly affect the application development landscape. As a result, the tools utilized in previous studies became outdated due to their inability to adjust to the constantly changing landscape of Android's development. Further emphasizing the necessity for a more approachable and straightforward solution in this area is the fact that none of the previous initiatives produced a freely accessible tool that is easy to use for the convenience of regular users.

## 3. SYSTEM DESIGN

### 3.1 System Architecture

The architecture of the system is depicted in Figure 1.

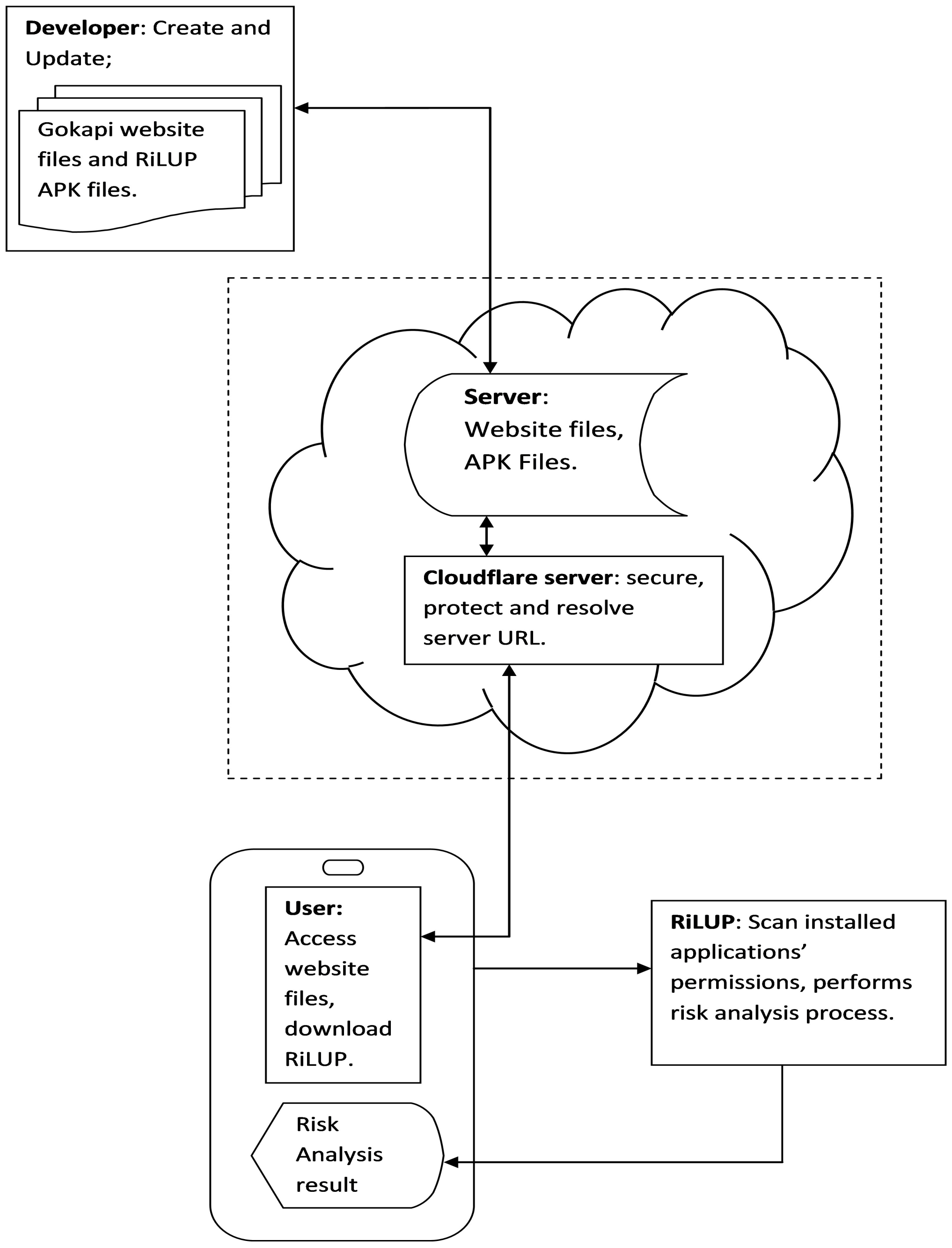

**Developer**: Create and Update;
Gokapi website files and RiLUP APK files.
**Server**: Website files, APK Files.
**Cloudflare server**: secure, protect and resolve server URL.
**User:** Access website files, download RiLUP.
**RiLUP**: Scan installed applications' permissions, performs risk analysis process.
Risk Analysis result

Figure 1: System Architecture

The developer created the website files, which aided in file sharing and the RiLUP files created on Android studio, used to develop the RiLUP application. These files were uploaded to the server. The website, which is powered by Gokapi, holds the RiLUP APK file which was uploaded to the website for easy access by the users. The server receives requests from the user and responds by sending the requested website files. The server is protected by Cloudflare so that when the DNS lookup for the website's URL, the server will resolve it to CloudflareAnycast IPs open external link instead of the server original name. The user's device runs Android operating system and contains several apps installed on it. The device accesses the website files and downloads the RiLUP application from the server. All installed apps are retrieved and their Android manifest.xml files searched to extract all permissions. Minifest.xml files contain information about each app and the permission it requests. The permissions extracted from each of the manifest.xml file are stored in a variable used by the risk algorithm, during the risk analysis process, to classify the risk level of each app as Low, Medium or High. The result of the risk analysis is displayed on the user's device in a way an average android user will understand.

## 3.2 Risk Algorithm

The risk algorithm used during the risk analysis process is depicted as follows:
Step 1: Check the Permissions Requested by an Application
Step 2: Identify all the Sensitive Permissions (S.P)
Step 3: Store all the S.P in the "sensitivePermissionNames" Variable
Step 4: Count the Total Number of S.P and Record as TSP
Step 5: Count all the S.P Granted by an Application and Record as TGSP
Step 6: Total S.P in RiLUP Application = TS (which is 19)
Step 7: Assign a Score of 2 to S.Ps that are Granted
Step 8: Assign a Score of 1 to S.Ps that are NOT Granted
Step 9: Calculate the Total Attainable Point (TAP) = TS * 3
Step 10: Calculate the Risk Score (R.S) = [(TGSP * 2) + TSP/ (TAP)]*100
Step 11: Classify the Risk: (Low: 0 – 25; Medium: 26 – 50; High: 51 – 100)

The 19 permissions declared as sensitive by the RiLUP application are

- android.Manifest.permission.CAMERA: This permission gives an application access to the device camera, and a photo or video can be taken without the user's consent.
- android.Manifest.permission.RECORD_AUDIO: This permission gives an application access to the device microphone and can record audio without the user's consent.
- android.Manifest.permission.READ_SMS: This permission is declared sensitive because it gives an application access to read through the messages in the Android device without the user's approval.
- android.Manifest.permission.WRITE_EXTERNAL_STORAGE: This permission gives an application the ability to write directly to external storage.
- android.Manifest.permission.READ_EXTERNAL_STORAGE: If a user grants this permission to an application, it has access to read the storage and deletes all its content.
- android.Manifest.permission.READ_PHONE_STATE: READ_PHONE_STATE is one of the Android permissions categorized as dangerous. This is because it "allows read-only access to phone's state, including the phone number of the device, current cellular network information, the status of any ongoing calls, and a list of any Phone Accounts registered on the device

- android.Manifest.permission.READ_CALL_LOG: This permission allows applications to save the device's call log data, and malicious applications may share call log data without the user's knowledge.
- android.Manifest.permission.WRITE_CALL_LOG: Allows the application to modify the phone's call log, including data about incoming and outgoing calls. Malicious applications may use this to erase or modify the device call log.
- android.Manifest.permission.SEND_SMS: This permission allows an application to send messages.
- android.Manifest.permission.RECEIVE_SMS: It allows the application to listen to all the SMSs that are received on the user's phone while he/she is using the application.
- android.Manifest.permission.CALL_PHONE: This permission allows an application to make a phone call without the user having to confirm it through the Dialer user interface.
- android.Manifest.permission.ACCESS_FINE_LOCATION: Allows the API to determine as precise a location as possible from the available location providers, including the Global Positioning System (GPS) as well as WiFi and mobile cell data.
- android.Manifest.permission.ACCESS_COARSE_LOCATION:
- android.Manifest.permission.READ_CONTACTS: Applications with this permission are able to save contact information, and malicious applications may do so without the user's awareness.
- android.Manifest.permission.WRITE_CONTACTS: This permission enables the application to change the information about contacts that is saved on the phone, including how often the user has called, emailed, or spoken to them in other ways. Using this permission, applications can remove contact information.
- android.Manifest.permission.READ_CALENDAR: This permission allows the application to read every calendar entry on the phone, including friends' and coworkers' activities. As a result, the application could be able to share or save calendar information regardless of its sensitivity or confidentiality.
- android.Manifest.permission.WRITE_CALENDAR: This permission allows the application to add, remove, amend, and edit events on the device, including friends' and coworkers' events. As a result, the application might be able to alter events without the owner's awareness or send messages that appear to be from the owner of the calendar.
- android.Manifest.permission.QUERY_ALL_PACKAGES: This permission gives application visibility into the inventory of installed applications on a given device
- android.Manifest.permission.MANAGE_EXTERNAL_STORAGE: An application can access all external storage in scoped storage with the MANAGE_EXTERNAL_STORAGE permission, intended for use by a select few applications that must manage files on the users' behalf

## 3.3 UML Class Diagram

The class diagram models the classes in the RiLUP code, their attributes/methods, and the relationship between them. The class diagram is represented in Figure 2.

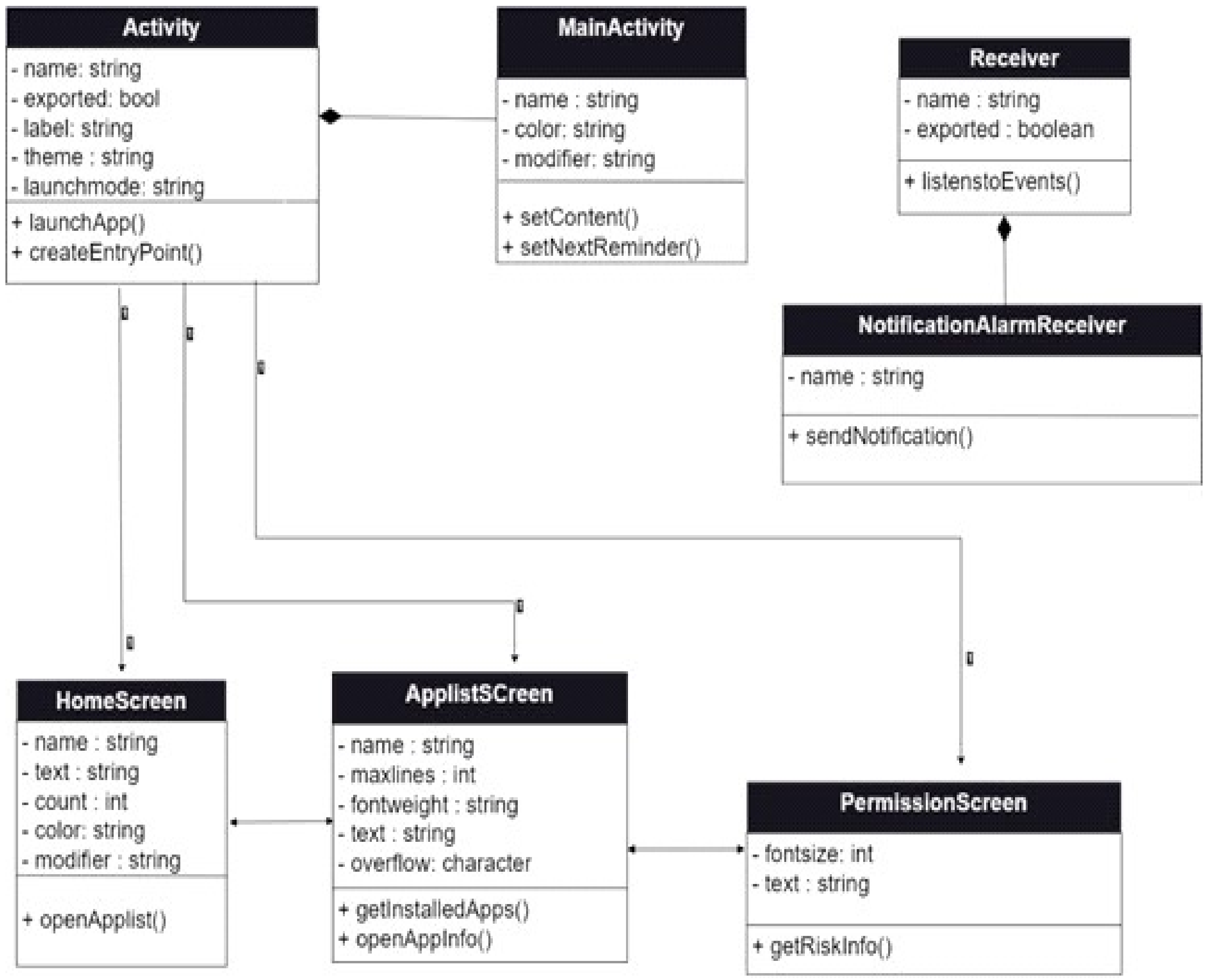


Figure 2: UML Class Diagram

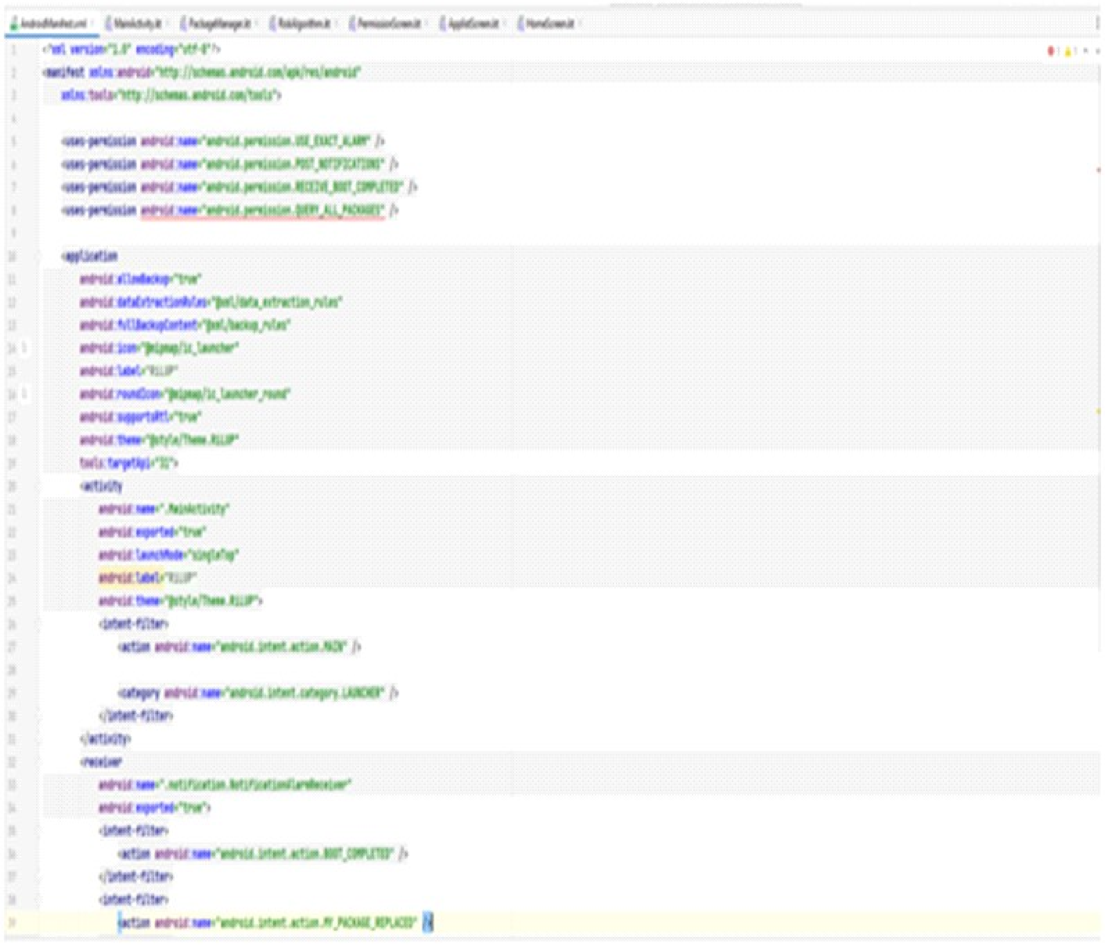
Activity class is the parent class to HomeScreen, PermissionScreen, and ApplistScreen classes. The MainActivity class has the methods setContent() and setNextReminder(), which define the design of the interface and respectively trigger the notification immediately one is sent. The receiver class is a system class that declares a broadcast receiver that enables the application to receive intents even when other components of the application are not running. The method, listenstoEvents(),carries out the intent of the broadcast receiver by listening to when an event occurs. It is a parent class to the NotificationAlarmReceiver class. The method, openApplist () in the HomeScreen class opens the next screen in the RiLUPapplication.The method getInstalledApp() in the ApplistScreen retrieves the applications installed on the device while the method getAppInfo() returns details about each application. The method getRiskInfo() in the PermissionScreen gets the permission requested by each application and count the number of permissions that are sensitive.

## 4. IMPLEMENTATION

The RiLUP application was developed using the Kotlin programming language and the Jetpack Compose UI framework. The development of some key components of RiLUPis illustrated in subsections 4.1 to 4.3.

### 4.1 Application Manifest File

This file is created when a new project is created on the Android Studio SDK. It contains the metadata (package name, activity names, main activity, Android version support, permissions, and other configurations) of RiLUP. The category for the intent filter is named android.intent.category.LAUNCHER to indicate that the application can be launched from the device's launcher icon. The RiLUP manifest file is represented in Figure 3.



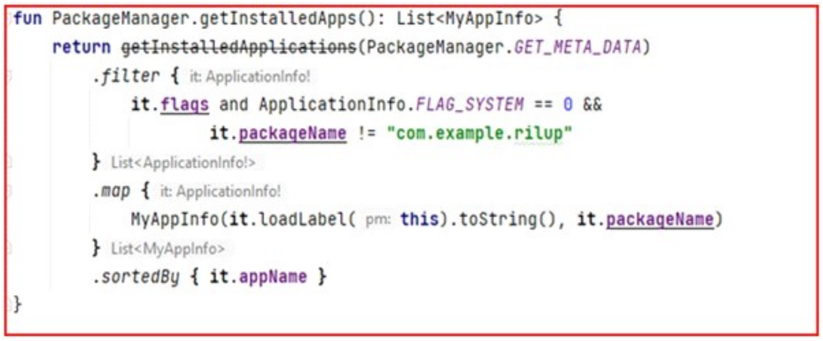
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-permission android:name="android.permission.USE_EXACT_ALARM" />
    <uses-permission android:name="android.permission.POST_NOTIFICATIONS" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <uses-permission android:name="android.permission.QUERY_ALL_PACKAGES" />

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="RiLUP"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.RiLUP"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:launchMode="singleTop"
            android:label="RiLUP"
            android:theme="@style/Theme.RiLUP">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <receiver
            android:name=".notification.NotificationAlarmReceiver"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.MY_PACKAGE_REPLACED" />
```

Figure 3: RiLUP Android Manifest File

- Package Manager

The packageManager.kt file contains all the code that works on applications. The packageManager class was used to retrieve various kinds of information related to the application packages that were installed on the device. This file contains the following components:

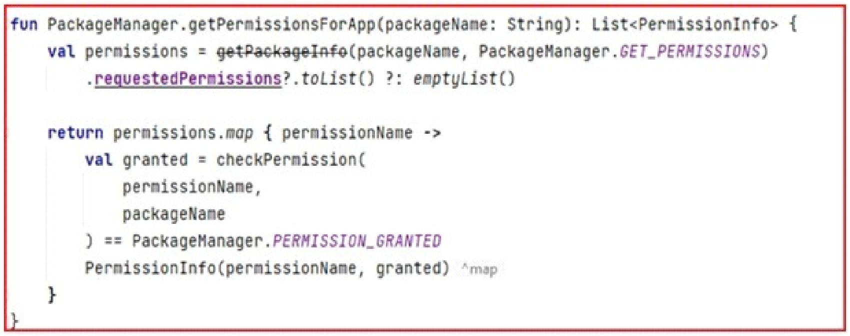
- getInstalledApps: This function, represented in Figure 4, works with the packageManagerclass and the method getInstalledApplication, which returns a list of all application packages that are installed on the Android device. The function removes system applications like contacts, phone dialers, etc. by filtering the returned list and removing applications with the flag FLAG_SYSTEM, which denotes that the application is installed in the device's system image. The function also uses the filter method to remove the RiLUP application if it is installed on the device because it requires the QUERY_ALL_PACKAGES permission, which allows the query of any application on the device. The list is then sorted by the application name and called in the ApplistScreen to display the installed applications.



```
fun PackageManager.getInstalledApps(): List<MyAppInfo> {
    return ~~getInstalledApplications~~(PackageManager.GET_META_DATA)
        .filter { it: ApplicationInfo!
            it.flags and ApplicationInfo.FLAG_SYSTEM == 0 &&
                    it.packageName != "com.example.rilup"
        } List<ApplicationInfo!>
        .map { it: ApplicationInfo!
            MyAppInfo(it.loadLabel( pm: this).toString(), it.packageName)
        } List<MyAppInfo>
        .sortedBy { it.appName }
}
```

Figure 4: getInstalledApps Function

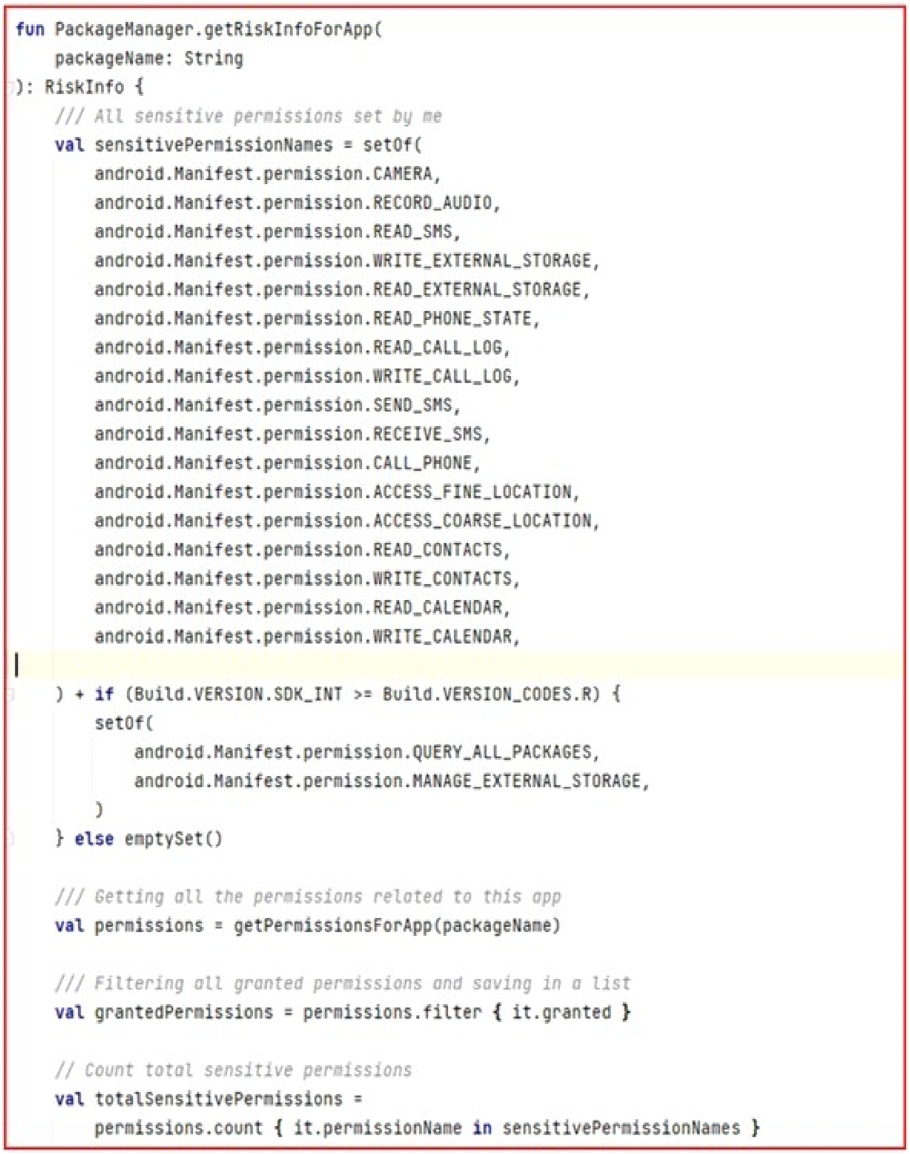
- getPermissionsForApp: This function calls the packageManager class and use the getPackageInfo method and the GET_PERMISSIONS flag, to return information about permissions in the application. It stores these permissions in a variable with name permissions, which is then mapped to get granted permissions using the PERMISSION_GRANTED method and stored in the variable named "granted". The getPermissionsForApp Function is represented in Figure 5.



```
fun PackageManager.getPermissionsForApp(packageName: String): List<PermissionInfo> {
    val permissions = ~~getPackageInfo~~(packageName, PackageManager.GET_PERMISSIONS)
        .requestedPermissions?.toList() ?: emptyList()

    return permissions.map { permissionName ->
        val granted = checkPermission(
            permissionName,
            packageName
        ) == PackageManager.PERMISSION_GRANTED
        PermissionInfo(permissionName, granted) ^map
    }
}
```

Figure 5: getPermissionsForApp Function

- **getRiskInfoForApp:** This function declares sensitive permissions and classifies the permission retrieved using different variables; these variables are declared with the keywords “val”, sensitivePermissionNames, permissions, grantedPermissions, totalSensitvePermissions, grantedSensitivePermissions, and score, which hold the values of permission information obtained from each application. The getRiskInfoForApp function is depicted in Figure 6.

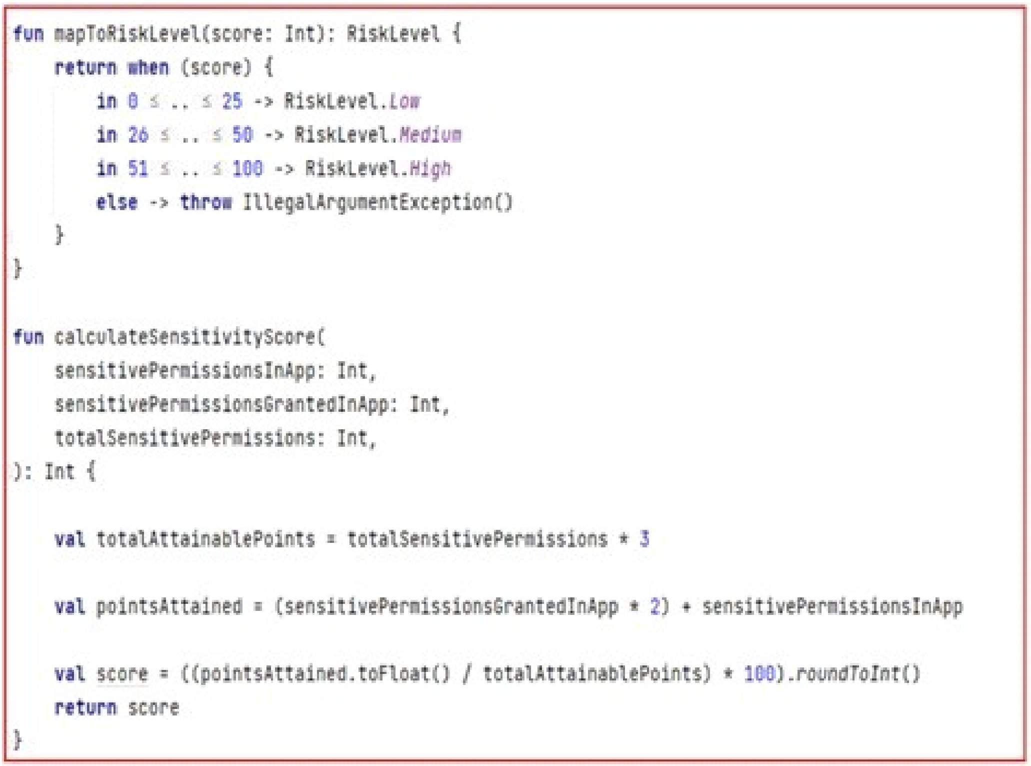


```
fun PackageManager.getRiskInfoForApp(
    packageName: String
): RiskInfo {
    /// All sensitive permissions set by me
    val sensitivePermissionNames = setOf(
        android.Manifest.permission.CAMERA,
        android.Manifest.permission.RECORD_AUDIO,
        android.Manifest.permission.READ_SMS,
        android.Manifest.permission.WRITE_EXTERNAL_STORAGE,
        android.Manifest.permission.READ_EXTERNAL_STORAGE,
        android.Manifest.permission.READ_PHONE_STATE,
        android.Manifest.permission.READ_CALL_LOG,
        android.Manifest.permission.WRITE_CALL_LOG,
        android.Manifest.permission.SEND_SMS,
        android.Manifest.permission.RECEIVE_SMS,
        android.Manifest.permission.CALL_PHONE,
        android.Manifest.permission.ACCESS_FINE_LOCATION,
        android.Manifest.permission.ACCESS_COARSE_LOCATION,
        android.Manifest.permission.READ_CONTACTS,
        android.Manifest.permission.WRITE_CONTACTS,
        android.Manifest.permission.READ_CALENDAR,
        android.Manifest.permission.WRITE_CALENDAR,

    ) + if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.R) {
        setOf(
            android.Manifest.permission.QUERY_ALL_PACKAGES,
            android.Manifest.permission.MANAGE_EXTERNAL_STORAGE,
        )
    } else emptySet()

    /// Getting all the permissions related to this app
    val permissions = getPermissionsForApp(packageName)

    /// Filtering all granted permissions and saving in a list
    val grantedPermissions = permissions.filter { it.granted }

    // Count total sensitive permissions
    val totalSensitivePermissions =
        permissions.count { it.permissionName in sensitivePermissionNames }
```

Figure 6: getRiskInfoForApp

- Risk ALgorithm.kt File

This file contains the code for the risk algorithm that calculated the risk score and the range used to map the risk score. The algorithm gives a risk score based on the number of sensitive permissions required and the number of sensitive permissions granted. The function to calculate the Risk Score is shown in Figure 7.



```
fun mapToRiskLevel(score: Int): RiskLevel {
    return when (score) {
        in 0 ≤ .. ≤ 25 -> RiskLevel.Low
        in 26 ≤ .. ≤ 50 -> RiskLevel.Medium
        in 51 ≤ .. ≤ 100 -> RiskLevel.High
        else -> throw IllegalArgumentException()
    }
}

fun calculateSensitivityScore(
    sensitivePermissionsInApp: Int,
    sensitivePermissionsGrantedInApp: Int,
    totalSensitivePermissions: Int,
): Int {

    val totalAttainablePoints = totalSensitivePermissions * 3

    val pointsAttained = (sensitivePermissionsGrantedInApp * 2) + sensitivePermissionsInApp

    val score = ((pointsAttained.toFloat() / totalAttainablePoints) * 100).roundToInt()
    return score
}
```

Figure 7: mapToRiskLevel and calculateSensitivityScore Function

## 4.4 Uploading RiLUP to Website

A domain name “olivianwose.dev” was bought from Name Cheap to host a site for easy sharing of the RiLUP APK file. The code for the site is open-source code from the GitHub repository. The RiLUP APK file was uploaded to the site, which creates a link that can be shared for the RiLUP to be downloaded. This site is controlled by Gokapi, a lightweight server to share files, which expire after a set amount of downloads or days. The server is protected by Cloudflare, which prevents unwanted access to the server's IP address. Cloudflare links the domain name system (DNS) to the server any time the domain name “olivianwose.dev” is searched on the internet.

## 4.5 Outputs

Some of the screen shots displayed while running the RiLUP app sis shown in Figures 8 – 11.

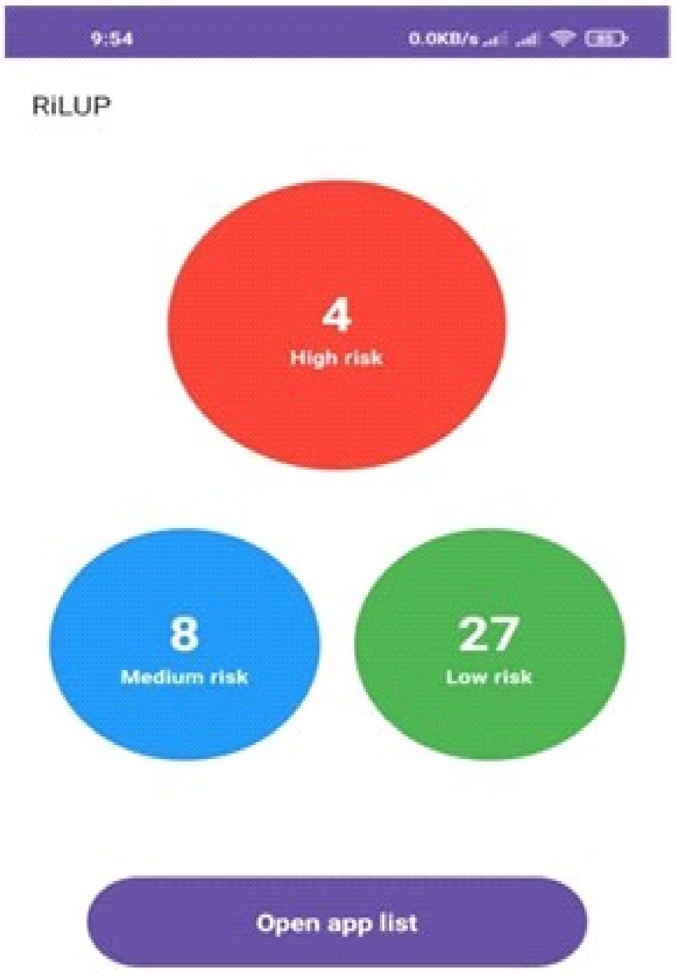
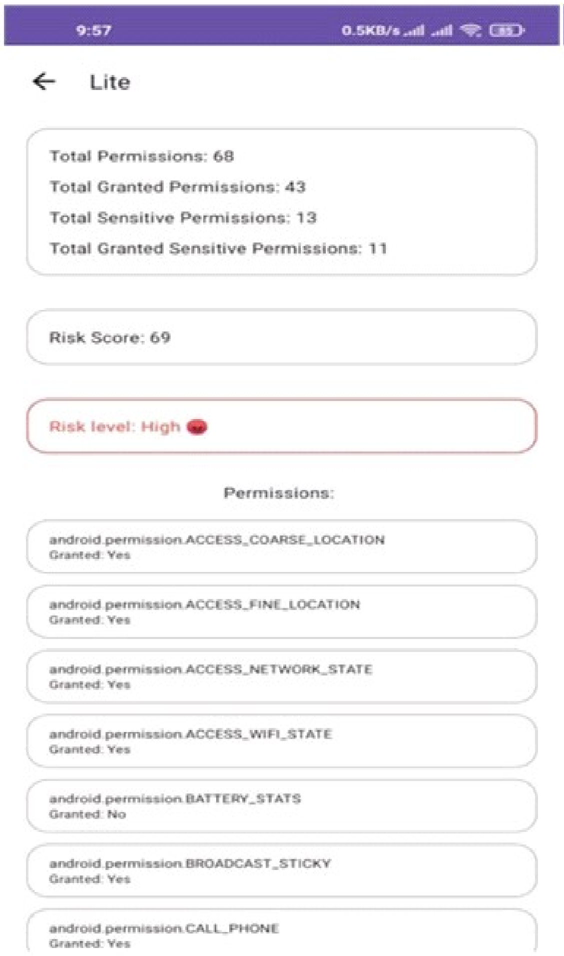


Figure 8: RiLUP Home Screen



9:57 0.5KB/s

Lite

Total Permissions: 68
Total Granted Permissions: 43
Total Sensitive Permissions: 13
Total Granted Sensitive Permissions: 11

Risk Score: 69

Risk level: High

Permissions:

android.permission.ACCESS_COARSE_LOCATION
Granted: Yes

android.permission.ACCESS_FINE_LOCATION
Granted: Yes

android.permission.ACCESS_NETWORK_STATE
Granted: Yes

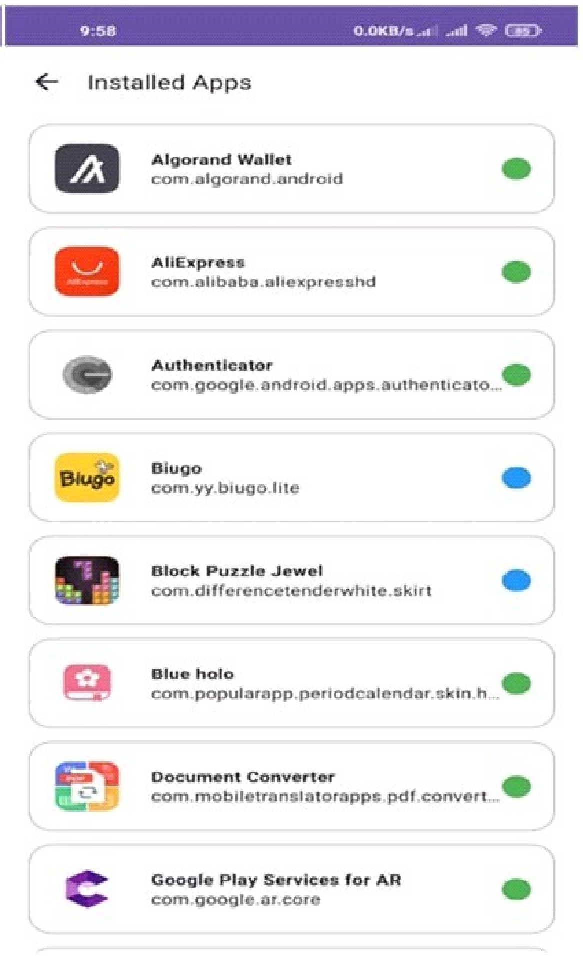
android.permission.ACCESS_WIFI_STATE
Granted: Yes

android.permission.BATTERY_STATS
Granted: No

android.permission.BROADCAST_STICKY
Granted: Yes

android.permission.CALL_PHONE
Granted: Yes

Fig. 9: RiLUPPermission Screen

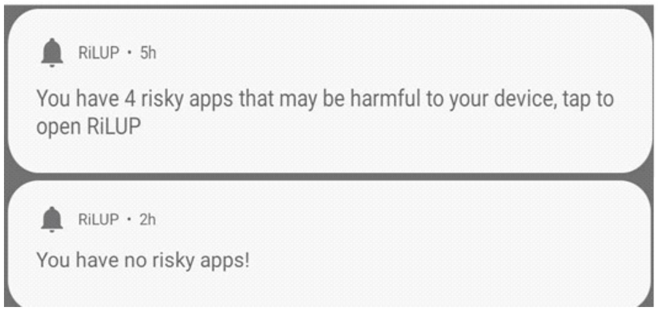


9:58 0.0KB/s

Installed Apps

| | |
|---|---|
| Algorand Wallet<br>com.algorand.android | 🟢 |
| AliExpress<br>com.alibaba.aliexpresshd | 🟢 |
| Authenticator<br>com.google.android.apps.authenticato... | 🟢 |
| Biugo<br>com.yy.biugo.lite | 🔵 |
| Block Puzzle Jewel<br>com.differencetenderwhite.skirt | 🔵 |
| Blue holo<br>com.popularapp.periodcalendar.skin.h... | 🟢 |
| Document Converter<br>com.mobiletranslatorapps.pdf.convert... | 🟢 |
| Google Play Services for AR<br>com.google.ar.core | 🟢 |

Fig. 10: RiLUP Installed Applications List Screen



RiLUP • 5h
You have 4 risky apps that may be harmful to your device, tap to open RiLUP

RiLUP • 2h
You have no risky apps!

Figure 11: RiLUP Notification PopUp

## 4.6 Evaluation and Results

RiLUP application was used to analyze the uses permissions of several Android applications and the corresponding risk score and level based on the permissions requested were generated; this was used to test and evaluate the developed application. Table 2 shows the list of 40 applications that were analyzed by RiLUP app.

| S/N | Application Name | TAP | TP | TGP | TSP | TGSP | Risk Score | Risk Level |
|---|---|---|---|---|---|---|---|---|
| 1 | Algorand Wallet | 51 | 17 | 11 | 6 | 0 | 12 | Low |
| 2 | AliExpress | 51 | 38 | 21 | 8 | 0 | 16 | Low |
| 3 | Audio Bible | 51 | 15 | 14 | 3 | 3 | 18 | Low |
| 4 | Authenticator | 51 | 12 | 12 | 1 | 1 | 6 | Low |
| 5 | Block Puzzle Jewel | 51 | 20 | 18 | 5 | 5 | 29 | Medium |
| 6 | Biugo | 51 | 51 | 19 | 6 | 0 | 12 | Medium |
| 7 | Block-Puzzle Jewel | 51 | 29 | 17 | 4 | 0 | 8 | Medium |
| 8 | Document Converter | 51 | 10 | 10 | 3 | 3 | 18 | Low |
| 9 | Google-Play | 51 | 6 | 5 | 0 | 0 | 0 | Low |
| 10 | Jigsaw Puzzles | 51 | 10 | 9 | 1 | 0 | 2 | Low |
| 11 | JulWallet | 51 | 26 | 11 | 3 | 0 | 6 | Low |
| 12 | Lite | 51 | 70 | 43 | 13 | 11 | 69 | High |
| 13 | Magic Tiles | 51 | 8 | 7 | 2 | 2 | 12 | Low |
| 14 | MetaMask | 51 | 26 | 10 | 2 | 0 | 6 | Low |
| 15 | Mi Doc Viewer | 51 | 13 | 10 | 2 | 2 | 12 | Low |
| 16 | Netflix | 51 | 18 | 12 | 2 | 0 | 6 | Low |
| 17 | Noizz | 51 | 52 | 25 | 6 | 6 | 35 | Medium |
| 18 | Phone Master | 51 | 50 | 31 | 7 | 7 | 41 | Medium |
| 19 | Pi | 51 | 29 | 12 | 1 | 0 | 2 | Low |
| 20 | Pi Browser | 51 | 32 | 16 | 5 | 5 | 29 | Medium |
| 21 | Roqqu | 51 | 12 | 12 | 4 | 4 | 24 | Low |
| 22 | Shazam | 51 | 24 | 17 | 4 | 2 | 24 | Low |
| 23 | Showmax | 51 | 16 | 12 | 1 | 0 | 2 | Low |
| 24 | Smart Puzzles | 51 | 7 | 7 | 0 | 0 | 0 | Low |
| 25 | Status-Story Saver | 51 | 31 | 10 | 3 | 0 | 9 | Low |
| 26 | Telegram | 51 | 56 | 25 | 11 | 2 | 29 | Medium |
| 27 | Tick | 51 | 43 | 23 | 9 | 9 | 53 | High |
| 28 | Tile Master 3D | 51 | 14 | 11 | 2 | 0 | 6 | Low |
| 29 | TimeJot | 51 | 2 | 2 | 0 | 0 | 0 | Low |
| 30 | Too Hot to Handle | 51 | 15 | 12 | 0 | 0 | 0 | Low |
| 31 | Truecaller | 51 | 82 | 52 | 15 | 15 | 88 | High |
| 32 | Trust Wallet | 51 | 12 | 8 | 2 | 0 | 6 | Low |
| 33 | VFly | 51 | 52 | 25 | 6 | 6 | 35 | Medium |
| 34 | VLC | 51 | 19 | 14 | 2 | 1 | 12 | Low |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 35 | WPS Office | 51 | 34 | 20 | 3 | 1 | 15 | Low |
| 36 | Wallpaper Carousel | 51 | 23 | 19 | 1 | 0 | 3 | Low |
| 37 | Water Color Sort | 51 | 11 | 9 | 1 | 0 | 3 | Low |
| 38 | WhatsApp Business | 51 | 69 | 44 | 7 | 3 | 39 | Medium |
| 39 | Xender | 51 | 164 | 31 | 10 | 10 | 59 | High |
| 40 | X | 51 | 50 | 26 | 8 | 1 | 18 | Low |

**KEY:** TAP - Total Attainable Points; TP - Total Permissions; TGP - Total Granted Permissions, TSP - Total Sensitive Permissions; TGSP - Total Granted Sensitive Permissions

The risk score is based on a point system where two points is assigned when sensitive permission is declared and granted; one point is assigned when it is declared but not granted. So 3 (2 + 1) is the total point for each sensitive permission. The Risk Score for each application is the percentage of the points attained by the application compared to the total attainable points (TAP) from all sensitive permissions (TS) available in the RiLUP framework. That is:

Risk Score = [(TGSP * 2) + TSP/ (TAP)]*100 (1)

*Where*
TAP = TS * 3 = 17*3 = 51. (2)

For example, the Risk Score for the first application in Table 2 is calculated thus:
[(0 * 2) + 6 / 51)] * 100= [6/51] * 100 = 0.1176 * 100 = 11.8 (Approximately 12)

## 5. CONCLUSION

The RiLUP application addresses the increasing importance of security and privacy in the digital era, particularly with the widespread use of smartphones and Android applications. This work develops software-based risk level analysis of uses permission on android platform which allows Android phone users manage permissions granted to installed applications.

The RiLUP application includes a robust risk rating system, providing users with an easily understandable risk score for each application based on its permissions. This score helps users to quickly assess potential risks quickly, making them aware of permissions they may have unknowingly granted. The application encourages informed decisions on retaining, revoking, or uninstalling applications. Designed for scalability, RiLUP is ready for future updates, including adjustments to the risk assessment algorithm or the addition of features based on user feedback. In conclusion, RiLUP represents a significant step towards a safer digital environment, acting as a vigilant guard to ensure users retain control over their personal data in an era where data security is crucial.

## REFERENCES

1. Alazab, M., Alazab, M., Shalaginov, A., Mesleh, A., and Awajan, A. (2020). Intelligent Mobile Malware Detection using Permission Requests and API calls. Future Generation Computer Systems, 107, 509-521. Doi:10.1016/j.future.2020.02.002.
2. Alenezi, M. andAlmomani, I. (2017). Abusing Android Permissions: A security Perspective. In Proceedings of 2017 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT), DOI:10.1109/AEECT.2017.8257772
3. Alshehri, A., Marcinek, P., Alzahrani, A., Alshahrani, H., and Fu, H. (2019). PUREDroid: Permission Usage and Risk Estimation for Android Applications. In proceedings of the 3rd International Conference on Information Systems and Data Mining, Houston TX, USA, April 6 - 8, 2019, 179–184. Doi.org/10.1145/3325917.332594
4. AppBrain (2024). Android Statistics: Number of Android Aps on Google Play. Http://www.appbrain.com/statistics/number-of-android-apps (visited, April 2024)
5. Ashawa, M. and Morris, S. (2021). Android Permission Classifier: A Deep Learning Algorithmic Framework based on Protection and Threat Levels. Security and Privacy 4(11), e164, 1 – 26 .Doi:10.1002/spy2.164
6. Developer (2023). Permissions on Android. https://developer.android.com/guide/topics/permissions/overview
7. Dini, G., Martinelli, F., Matteucci, I., Petrocchi, M., Saracino, A., and Sgandurra, D. (2018). Risk Analysis of Android Applications: A User-centric Solution. Future Generation Computer Systems, 80, 505-518. Doi:10.1016/j.future.2016.035.
8. Guaman, D., Del Alamo, J. and Caiza, J. (2021). GDPR Compliance Assessment for Cross-Border Personal Data Transfers in Android Apps. IEEE Access, Volume 9, 15961 – 15982. Doi: 10.1109/ACCESS.2021.3053130
9. Huang, K., Zhang, J., Tan, W., Feng, Z. (2015). An Empirical Analysis of Contemporary Android Mobile Vulnerability Market. In Proceeding of 2015 IEEE International Conference on Mobile Services, New York, USA, 182–189. DOI: 10.1109/MobServ.2015.34
10. Olukoya O., Mackenzie L., and Omoronyia I. (2019). Security-oriented View of App Behaviour using Textual Descriptions and User-granted Permission Requests. Computers and Security, Volume 89, https://doi.org/10.1016/j.cose.2019.101685
11. Shen, Y., Xu, M., Zheng, N., Xu, J., Xia, W., Wu, Y., Qiao, T., and Yang, T. (2018). Android application classification and permission usage risk assessment. In: Romdhani, I., Shu, L., Takahiro, H., Zhou, Z., Gordon, T., Zeng, D. (eds) Collaborative Computing: Networking, Applications and Worksharing. CollaborateCom 2017. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 252. Springer, Cham. https://doi.org/10.1007/978-3-030-00916-8_52
12. Stegner, B. (2023). How do Android App permissions work? What you need to Know.
13. https://www.makeuseof.com/tag/what-are-android-permissions-why-should-you-care/(Visited April, 2024)
14. Wang, Y., Zheng, J., Sun, C. and Mukkamala, S. (2013). Quantitative Security Risk Assessment of Android Permissions and Applications. In: Wang, L., Shafiq, B. (eds) Data and Applications Security and Privacy XXVII. DBSec 2013. Lecture Notes in Computer Science, vol 7964. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-39256-6_15

15. Wu, S., and Liu, J. (2019). Overprivileged Permission Detection for Android Applications. In Proceedings of IEEE International Conference on Communications (ICC), Shanghai, China, May 20 – 24, 2019. Doi:10.1109/icc.2019.8761572.
16. Xiao, J., Chen. S., He, Q., Feng, Z. and Xwe, X. (2020). An Android Application Risk Evaluation Framework Based on Minimum Permission set Identification. Journal of Systems and Software, 163, 110533, 1 – 17
17. Yilmaz, S. and Davis,M. (2023). Hidden Permissions on Android: A Permission-Based Android Mobile Privacy Risk Model. In Proceedings of the 22nd European Conference on Cyber warfare and Security (ECOWS, 2023), 717 – 724. Doi:10.34190/eccws.22.1.1453
18. Zhang, Y., Yang, M., Gu, G. , Chen, H. (2016). Rethinking Permission Enforcement Mechanism on Mobile Systems. IEEE Transactions on Information Forensics and Security 11 (10), 2227–2240.