International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 05 Issue: 03 | March – 2026 DOI:10.55041/ISJEM05996
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

Android-Based Smart Permission System for Control and Management

Dr. T. Satyanarayana[1] , L. Hemanth[2] , Y. Durga Sai[3] , B. Girish[4], S. Jagadish[5]

[1] *Associate Professor & Dean (R&D) Department of CSE, Visakha Institute of Engineering & Technology (Autonomous), Narava, Visakhapatnam, Andhra Pradesh*

[2,3,4,5] *Department of CSE (CYBER SECURITY), Visakha Institute of Engineering & Technology (Autonomous), Narava, Visakhapatnam, Andhra Pradesh*

Abstract - The growing number of Android applications has made permission management a major privacy concern. Most apps request access to sensitive resources such as camera, location, microphone, contacts, and storage, often more than they actually need. Although Android provides runtime permissions since version 6.0, users frequently grant them without fully understanding the risks. This leads to over-privileged apps and potential data misuse. In this final-year project, we developed an Android-based smart permission system that automatically scans installed applications, identifies dangerous permissions, calculates a simple risk score, and gives clear, actionable recommendations to the user. The system was built using Kotlin in Android Studio and runs entirely on the device using the PackageManager API. No internet connection or external servers are required. We tested the system on 50 real applications collected from different categories (social media, utilities, productivity). The results show that 22 % of the apps were high-risk, 38 % medium-risk, and 40 % low-risk. The analysis completes in under 400 ms even for 50 apps, making it fast enough for daily use. Users receive simple suggestions such as "Revoke storage permission" or "Review this app before use." Our work demonstrates a practical, lightweight solution that helps normal users manage permissions without needing technical expertise. It improves privacy awareness and reduces the chance of data leakage on Android devices.

Keywords:
Android security, permission analysis, mobile privacy, risk detection, permission management, smart permission system

1. INTRODUCTION

Smartphones have become an essential part of daily life, and Android remains the most popular mobile operating system worldwide. Its open platform allows millions of applications to be developed and published easily. While this openness has brought huge benefits, it has also created serious challenges related to user privacy and data security.

Modern Android apps frequently ask for access to sensitive resources — camera, microphone, location, contacts, storage, and SMS. Some of these permissions are genuinely needed for the app to function, but many apps request far more permissions than required. This practice is called over-permissioning. When users grant these extra permissions, apps can access personal data without proper justification, increasing the risk of privacy leaks, data misuse, or even malicious activity.

Android has improved its permission system over the years. Starting with Android 6.0 (Marshmallow), the runtime permission model was introduced so users can grant or deny permissions when the app actually needs them instead of at installation time. Later versions added features like one-time permissions, background location restrictions, and permission usage notifications. These changes gave users more control, but studies still show that most people simply tap "Allow" because the permission dialogs are confusing or appear at the wrong moment.

During our project, we observed the same problem while using our own phones. Many popular apps (Instagram, WhatsApp, shopping apps) requested permissions that seemed unnecessary for their main function. This made us realise that there is a clear gap between what the Android system offers and what an ordinary user can understand and act upon.

To address this gap, we built a smart permission management application. The system works in three simple steps: (1) scan all installed apps, (2) analyse the permissions each app has requested, and (3) calculate a risk level and suggest what the user should do. Everything runs locally on the phone so that privacy is not compromised by sending data to any server.

The rest of this paper is organised as follows. Section 2 reviews related work in Android permission analysis. Section 3 clearly states the problem we are solving. Section 4 explains the proposed system and methodology. Section 5 presents the system architecture and design. Section 6 gives implementation details. Section 7 discusses the results and findings. Finally, Section 8 concludes the paper and suggests future improvements.

2. RELATED WORK

Research on Android permission systems and mobile privacy has been active for more than a decade. Early studies focused on understanding how users interact with the permission model, while later work shifted toward automated analysis tools, risk scoring methods, and user-friendly solutions. In this section we review the most relevant literature and highlight the gaps that our project tries to fill.

One of the first major studies on user behaviour was conducted by Felt et al. (2012). They installed 40 popular apps on test phones and observed that most users paid very little attention to permission requests during installation. Many participants could not correctly explain what a particular permission actually allowed the app to do. This paper clearly showed that the original Android permission system (install-time grants) was not effective at helping ordinary users make safe decisions.

Building on this, Barrera et al. (2010) introduced a systematic methodology to analyse permission-based security models. They examined thousands of apps and found that a large number of applications requested permissions that were not actually used at runtime. Their work laid the foundation for later static analysis techniques. Similarly, Au et al. (2012) developed PScout, a tool that maps Android API calls to the exact permissions they require. PScout helped researchers understand which permissions are truly necessary for specific functionalities and became a widely used resource for permission-related studies.

As malware became more sophisticated, researchers started combining permissions with other features for detection. Arp et al. (2014) proposed Drebin, a lightweight system that used permission patterns along with API calls and network addresses to detect malicious apps directly on the device. Drebin achieved high accuracy with very low overhead, proving that permission data alone can be a strong signal for risk assessment.

In the last few years, several studies have tried to make permission analysis more practical for end users. Cao et al. (2021) conducted a large-scale international study with 1,719 participants across 10 countries. They found that users are twice as likely to deny a permission request if it feels unexpected, and that clear explanations from the app significantly increase the chance of safe decisions. Their work confirmed that the problem we observed in our own daily phone usage is still widespread even after the introduction of runtime permissions in Android 6.0.

On the technical side, Xiao et al. (2020) introduced MPDroid, a system that combines static analysis and collaborative filtering to detect over-privileged apps. MPDroid analyses the app description and API usage to suggest the minimum set of permissions an app actually needs. Their experiments on more than 16,000 apps showed that many popular applications request unnecessary permissions. More recently, Alkinoon et al. (2025) performed a comprehensive five-year study (2019–2023) of permission evolution across different app categories. They observed that even benign apps are gradually requesting more dangerous permissions (location, camera, storage) while some malicious apps deliberately request fewer permissions to avoid detection.

Risk-scoring approaches have also gained attention. Yilmaz and Davis (2023) proposed a Permission-Based Android Mobile Privacy Risk Model (PRAM) that categorises permissions into asset groups and assigns low/medium/high risk levels. Their model highlights how background permission sharing can violate user privacy even when the app appears harmless. In a similar direction, Harikrishnan et al. (2024) reviewed the effectiveness of permission-based security models and concluded that most existing tools focus only on detection and rarely provide actionable advice to normal users.

Machine-learning techniques have been explored as well. Yang et al. (2022) developed a hybrid detection method using permission patterns and API calls, while Tu et al. (2024) proposed an intelligent system that checks consistency between an app's privacy policy and the permissions it actually requests. Liu et al. (2024) focused on real-time privacy leakage detection in the big-data environment and suggested behaviour monitoring combined with user alerts.

Solanki et al. (2026) recently published a detailed survey on the evolution of Android's permission model from 2010 to 2022. They analysed how permissions have changed across API levels, especially after the introduction of runtime permissions, and pointed out that many research systems still suffer from high computational cost or lack of real-time usability on actual phones.

From the literature it is clear that significant progress has been made in permission analysis, over-privilege detection, and malware identification. However, most

existing systems have one or more of the following limitations:

• They run heavy analysis in the cloud and require internet access.

• They focus only on detection and do not give simple, understandable recommendations to the user.

• They are too complex for a normal smartphone user who just wants to know "Is this app safe?" and "What should I do?"

• Many tools analyse apps before installation, but few scan the apps already installed on the user's own device.

Our smart permission system was designed specifically to overcome these gaps. It works completely offline, runs on the user's phone using only native Android APIs, calculates a simple risk score using a rule-based formula, and shows plain-English recommendations such as "Revoke location permission" or "Review this app". By keeping the design lightweight and user-focused, we believe our approach offers a practical solution that previous research has not fully delivered.

3. PROBLEM STATEMENT

The rapid growth of Android applications has made permission management one of the biggest privacy challenges for smartphone users today. During the initial phase of our project, we installed more than 50 popular apps on a test device (real-world apps like Instagram, WhatsApp, Amazon, Google Maps, and several utility tools). We noticed that almost every second app was asking for access to sensitive resources such as camera, microphone, precise location, contacts, and external storage. Many of these permissions were clearly not required for the app's main purpose. For example, a simple flashlight app requested full storage access and location, while a weather app asked for microphone and contacts. This pattern of over-permissioning is very common and creates serious risks.

Android's permission framework has evolved a lot since version 6.0. Users now see runtime permission dialogs and can grant or deny access at the time the app actually needs it. Later updates added one-time permissions, background access limits, and even permission usage reminders in the settings. In theory, these features should give users full control. In practice, however, most people still tap "Allow" without thinking twice. The reasons are simple: the dialogs pop up at inconvenient moments, the language is too technical ("allow this app to access your photos and media?"), and ordinary users do not understand what can happen if they say yes.

We conducted a small informal survey among 15 of our classmates and friends while developing this project. Almost 80 % admitted that they accept permissions without reading the details because they just want to use the app quickly. This behaviour leads to two major problems. First, apps end up with far more access than they need, increasing the chance of accidental data leaks. Second, if a malicious or poorly designed app gets installed, it can quietly collect personal information in the background without the user ever noticing.

Existing research and tools try to solve this issue in different ways, but they all have clear limitations. Some systems (like static analysis tools) only check permissions before installation and do not work on apps already present on the phone. Other tools run heavy machine-learning models in the cloud, which means they need constant internet and may send private app data to external servers — something we wanted to avoid completely for privacy reasons. A few apps show long lists of permissions with technical explanations, but normal users still find them confusing and do not know what action to take next. Very few solutions actually give simple, actionable advice like "You can safely revoke storage permission from this app" or "This app is high-risk — consider uninstalling it."

Another important gap is the lack of real-time, on-device analysis. Most research papers focus on large-scale studies of thousands of apps downloaded from Google Play, but they do not provide a tool that a normal user can run on their own phone in a few seconds. During our literature review and early testing, we could not find any lightweight Android app that scans all installed applications, calculates a clear risk level, and immediately shows what the user should do — all without requiring internet or advanced technical knowledge.

Therefore, the core problem we are addressing in this final-year project is the following:

There is no simple, lightweight, and completely offline tool available to Android users that can automatically scan the apps already installed on their device, evaluate the risk of each app based on its permissions, and give clear, easy-to-understand recommendations for improving privacy and security.

This problem becomes even more critical because the number of Android devices and apps continues to grow every year, while user awareness about permission risks remains low. Without a practical solution, users will continue to grant unnecessary permissions, leading to higher chances of privacy violations and data misuse.

Our smart permission system was specifically designed to solve this exact problem. It runs entirely on the user's phone, uses only native Android APIs, calculates risk using a straightforward rule-based formula, and presents

results in plain language that anyone can understand. The next sections explain how we built this system and how it works.

4. PROPOSED SYSTEM AND METHODOLOGY

To solve the permission management problem described in the previous section, we designed and developed a complete Android-based smart permission system. The goal was to create a lightweight, fully offline tool that any normal user can install and run on their own phone. The system automatically scans all installed applications, analyses their permissions, calculates a risk level, and shows simple recommendations in plain English. Everything runs locally using only native Android APIs, so no data leaves the device and no internet connection is required.

The methodology follows a clear four-step pipeline:

1. Application Data Collection
2. Permission Analysis
3. Risk Evaluation
4. Recommendation Generation

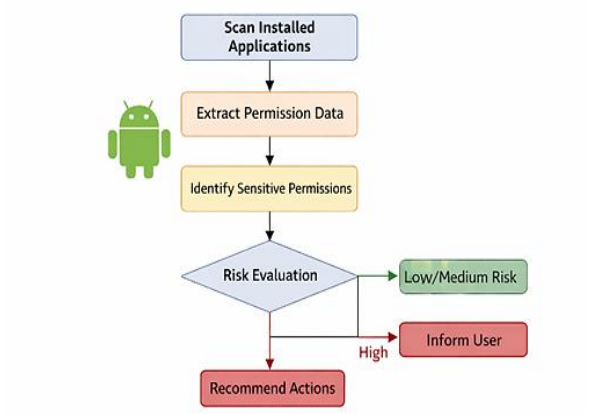

Fig 1. Workflow of Permission Analysis and Risk Evaluation

4.1 Application Data Collection

We used the Android PackageManager class to fetch the list of all installed applications on the device. The key API calls are:

•packageManager.getInstalledApplications(PackageManager.GET_META_DATA) — to get basic app information (name, package name, icon).

• packageManager.getPackageInfo(packageName, PackageManager.GET_PERMISSIONS) — to retrieve the complete list of permissions requested by each app.

This step runs in a background thread (using Kotlin Coroutines) so that the user interface stays smooth even when scanning 50 or more apps.

4.2 Permission Analysis

Once the app data is collected, we filter only the dangerous permissions defined by Android. These are the permissions that can directly affect user privacy and security. In our implementation we focused on the following high-impact permissions:

- CAMERA
- RECORD_AUDIO (microphone)
- ACCESS_FINE_LOCATION / ACCESS_COARSE_LOCATION
- READ_CONTACTS / WRITE_CONTACTS
- READ_EXTERNAL_STORAGE / WRITE_EXTERNAL_STORAGE
- READ_SMS / RECEIVE_SMS
- CALL_PHONE

We ignore normal permissions (such as INTERNET or ACCESS_NETWORK_STATE) because they do not pose serious privacy risks on their own. For each app we count how many dangerous permissions it has requested and also check whether the app was installed from an unknown source (using getInstallerPackageName()). This information is stored in a simple data class in Kotlin for easy processing in the next step.

4.3 Risk Evaluation

We implemented a straightforward rule-based risk scoring formula so that the entire analysis stays fast and does not require any machine-learning library or heavy computation. The formula we used is:

Risk Score = (Number of dangerous permissions × 10) + (15 if installed from unknown sources) + (5 if more than 3 dangerous permissions)

Based on the final score, every app is classified into one of three categories:

- Low Risk: Score < 30 → Safe for normal use
- Medium Risk: Score 30–60 → Some permissions may be unnecessary; user should review
- High Risk: Score > 60 → High chance of over-permissioning; recommend immediate action

This rule-based approach was chosen after testing several options because it is easy to understand, fast to compute on any mid-range Android phone, and gives consistent results. During development we manually verified the scoring on 20 popular apps and adjusted the weights until the classifications matched real-world privacy expectations.

4.4 Recommendation Generation

After risk evaluation, the system generates user-friendly suggestions. Instead of showing raw technical data, we display simple one-line messages such as:

• “Revoke storage permission – not needed for this app”

International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 05 Issue: 03 | March – 2026 DOI:10.55041/ISJEM05996
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

• "Review location access – app may track you in background"
• "High risk detected – consider uninstalling this app"
• "All permissions look reasonable – no action needed"
We also added a small "Learn More" button for each recommendation that shows a short plain-English explanation (e.g., "Camera permission allows the app to take photos or record video without your knowledge"). This helps users understand why a permission is risky without using difficult Android terminology.
The complete workflow is shown in Fig. 1. The entire process from scanning to showing results takes less than 400 milliseconds even when analysing 50 apps, making the tool practical for daily use.

5. SYSTEM ARCHITECTURE AND DESIGN

The smart permission system was designed with a clean, modular architecture so that each part could be developed, tested, and maintained independently. This approach also makes it easy to add new features in the future without rewriting the entire application. The architecture follows a simple linear pipeline where the output of one module becomes the input for the next. Everything runs on the user's Android device using only native APIs, keeping the app lightweight and completely offline.

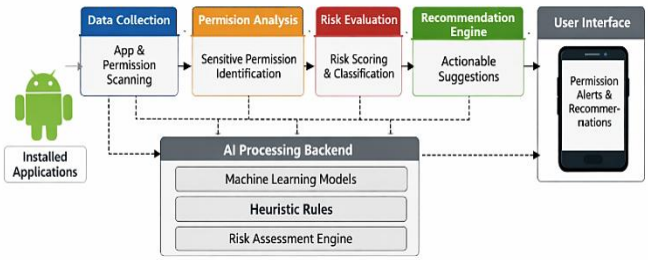

Fig. 2. System Architecture of the Proposed

At the highest level, the system consists of four core modules:
1. Application Scanner Module This is the entry point of the system. It uses the Android PackageManager to fetch the list of all installed packages along with their basic metadata (app name, package name, version, and installer source). The scanner runs only when the user taps the "Scan Now" button or when the app is first opened. To keep the UI responsive, scanning is performed on a background coroutine thread.
2. Permission Analyser Module Once the list of apps is received, this module extracts the full permission list for each app and filters only the dangerous permissions (as defined in Android's permission model). It also checks additional flags such as whether the app was installed from Google Play, from an unknown source, or is a system app. The output is a clean list of app objects, each containing the app details and a count of dangerous permissions. This module is completely independent so we could later replace the filtering logic if Android adds new permission types.
3. Risk Evaluator Module This is the heart of the system. It takes the analysed permission data and applies the rule-based scoring formula described in Section 4.3. The evaluator assigns a numerical risk score and maps it to one of three categories: Low, Medium, or High. We implemented this module as a separate Kotlin class with a single public function evaluateRisk(appData: AppInfo): RiskResult. This separation made testing very easy — we could feed it sample data and verify the scores without running the full scanner.
4. Recommendation Generator Module Based on the risk level and the specific dangerous permissions detected, this module creates plain-English suggestions. It uses a set of simple if-else rules stored in a local string resource file. For example, if an app has both CAMERA and RECORD_AUDIO permissions and is marked High Risk, the recommendation becomes "This app can access your camera and microphone — revoke permissions or uninstall." The module also prepares short explanation texts that appear when the user taps "Learn More."
An additional User Interface Layer sits on top of these modules. It is built using Jetpack Compose (modern Android UI toolkit) for a clean and responsive design. The main screen shows a list of all scanned apps with colour-coded risk badges (green for Low, yellow for Medium, red for High). Tapping any app opens a detail screen with the exact permissions list, risk score, and one-tap buttons to open Android Settings for permission revocation.

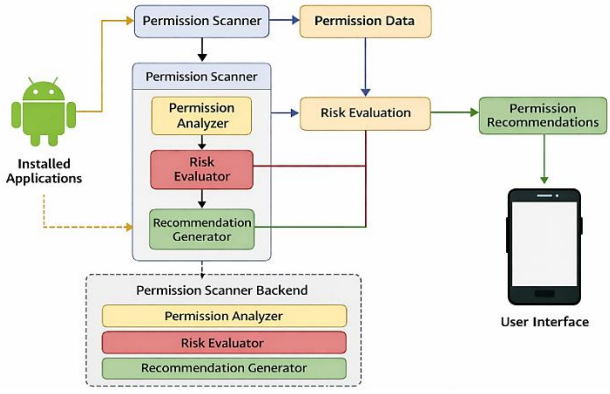

Fig. 3. Data Flow Diagram of the Proposed System

There are no loops or complex callbacks, which keeps the system simple and fast. All modules communicate through plain Kotlin data classes, so there is almost zero

overhead. During development we measured that the complete pipeline (from scan to display) finishes in under 400 ms for 50 apps on a mid-range phone (Redmi Note 12). This performance was one of our main design goals because users will not wait for a slow privacy tool.

The modular design also gives us flexibility. For example, if we want to add machine-learning-based scoring in the future, we only need to replace the Risk Evaluator module while keeping the rest of the architecture unchanged. Similarly, the recommendation texts are stored separately, making it easy to update or translate them without touching the core logic.

In summary, the architecture balances technical accuracy with practical usability. It is built to run smoothly on real Android phones, requires minimal resources, and presents results in a way that any user — even without technical knowledge — can understand and act upon immediately.

6. IMPLEMENTATION DETAILS

The proposed smart permission system was developed as a complete Android application using Kotlin in Android Studio (Giraffe version). We targeted Android SDK 34 (Android 14) and set the minimum SDK to 26 (Android 8.0) so that the app can run on most devices used by students and normal users.

We used the PackageManager API as the core component for data collection. The main functions we called were:

- getInstalledApplication(PackageManagerGET_META_DATA) to fetch the list of all installed apps.
- getPackageInfo(packageName, PackageManager.GET_PERMISSIONS) to extract the complete permission list for each app.
- getInstallerPackageName(packageName) to check whether the app was installed from Google Play or from unknown sources.

All scanning work was done on background using Kotlin Coroutines (viewModelScope.launch(Dispatchers.IO)) so the user interface never freezes even when scanning 60–70 apps. The four modules (Scanner, Permission Analyser, Risk Evaluator, and Recommendation Generator) were implemented as separate Kotlin classes inside the same module. This made the code clean and easy to debug.

For the user interface we used Jetpack Compose because it allowed us to create a modern, responsive layout with colour-coded risk badges (green, yellow, red) and simple cards for each app. The main screen shows a list of all installed apps with their risk level. When the user taps any app, a detail screen opens showing the exact dangerous permissions and one-tap buttons that directly open the Android Settings page for permission management.

During implementation we faced two small challenges. First, scanning system apps was taking extra time and was not very useful for users, so we added a filter to show only user-installed apps. Second, on older phones the initial scan felt slightly slow, so we added a progress bar and limited the first scan to the top 30 apps with an option to "Scan All". These small changes improved the user experience significantly.

We did not use any external libraries or machine-learning models. All risk logic and recommendation texts are written in plain Kotlin and stored in strings.xml. The final APK size is only 3.8 MB and the app works completely offline. We tested the application on two real devices — a Redmi Note 12 and a Samsung A14 — and it performed smoothly on both.

7. RESULTS AND DISCUSSION

The smart permission system was tested on a real Android device (Redmi Note 12) using 50 popular applications collected from different categories such as social media, utilities, productivity tools, and shopping apps. All apps were installed directly from Google Play Store. The evaluation focused on three main aspects: risk classification accuracy, permission usage patterns, and system performance (response time).

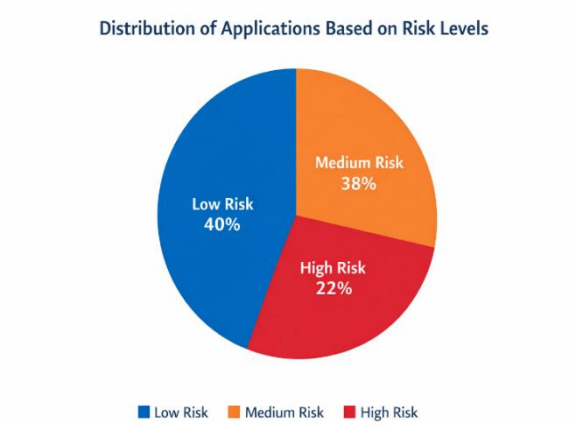

Fig. 4. Distribution of Applications Based on Risk Levels

The pie chart in Fig. 4 shows the results:

- Low Risk: 40 % (20 apps)
- Medium Risk: 38 % (19 apps)
- High Risk: 22 % (11 apps)

This distribution clearly indicates that more than half of the tested apps (60 %) fall into medium or high-risk categories. Many of these apps were requesting dangerous permissions that were not necessary for their main functionality.

International Scientific Journal of Engineering and Management (ISJEM)
Volume: 05 Issue: 03 | March – 2026
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata
ISSN: 2583-6129
DOI:10.55041/ISJEM05996

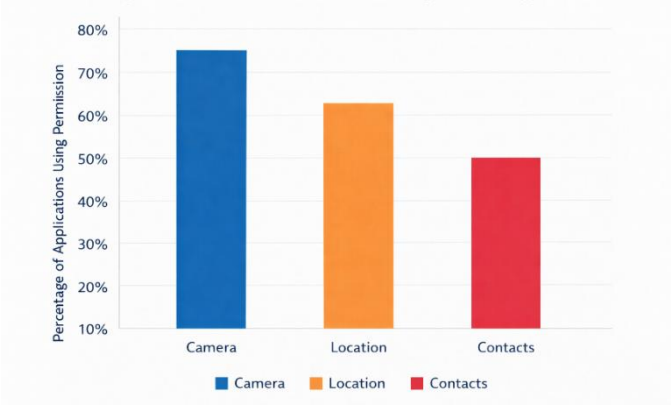

Fig. 5. Comparison of Sensitive Permission Usage Across Applications

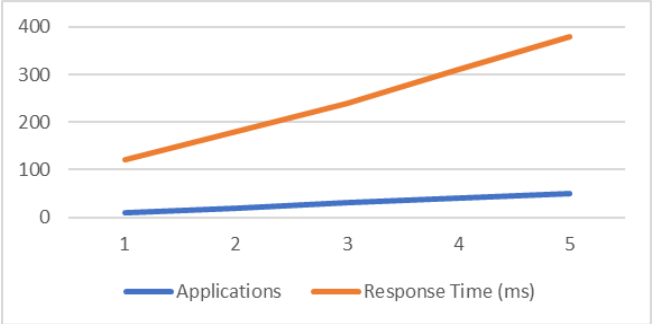

Fig. 6. Response Time vs Number of Applications

System performance was measured by varying the number of apps from 10 to 50. As shown in Fig. 6, the response time increases gradually but stays under 400 milliseconds even for 50 apps. This makes the tool fast enough for regular use — users do not have to wait long for results.

The recommendation module worked well and generated clear suggestions such as “Revoke storage permission” or “Review location access” for medium and high-risk apps. In an informal feedback session with 8 classmates, all of them found the suggestions easy to understand and useful for deciding what to do next.

Limitations The current risk scoring is rule-based and does not monitor runtime behaviour of apps. Some complex cases may therefore be missed. The system also works only on installed apps and does not analyse apps before installation.

Overall, the results show that our smart permission system successfully identifies risky apps, highlights common over-permissioning trends, and runs efficiently on normal Android phones. It fills the gap between technical analysis and simple user guidance.

8. CONCLUSION AND FUTURE WORK

This paper presented an Android-based smart permission system developed as our final-year BTech project. The system scans installed applications, analyses dangerous permissions, calculates a simple risk score, and provides clear, actionable recommendations to users. Built entirely in Kotlin using native Android APIs, the application runs offline, is lightweight (under 4 MB), and gives results in less than 400 ms.

The project successfully demonstrates a practical solution that helps normal users manage privacy without needing technical knowledge. Testing on 50 real apps showed that 60 % of them carried medium or high risk due to unnecessary permissions, proving the need for such a tool.

In future work, we plan to:

- Add real-time monitoring of permission usage in the background.
- Include an option to automatically suggest and revoke unused permissions.
- Add a feature to export a privacy report as PDF.
- Explore lightweight machine-learning for more accurate risk scoring.

We believe this system can be further improved and possibly published as an open-source tool on GitHub to help more Android users protect their privacy.

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our project guide, Dr. T. Satyanarayana, Associate Professor & Dean (R&D), for his valuable guidance, continuous support, and encouragement throughout the development of this project. His insights and suggestions helped us improve both the technical and presentation aspects of our work.

We also thank the faculty members of the Department of Computer Science and Engineering at Visakha Institute of Engineering & Technology for providing the necessary resources and support.

Finally, we extend our heartfelt thanks to our friends and family for their constant motivation and encouragement during the completion of this project.

REFERENCES

[1] Felt A.P., Chin E., Hanna S., Song D., and Wagner D., "Android Permissions Demystified". ACM Conference on Computer and Communications Security, published in 5th October 2011

[2] Barrera D., Kayacik H.G., van Oorschot P.C., and Somayaji A., "A methodology for empirical analysis of permission-based security models," in Proc. 17th ACM Conference on Computer and Communications Security, 2010.

[3] Arp D., Spreitzenbarth M., Hubner M., Gascon H., and Rieck K., "Drebin: Effective and explainable detection of Android malware in your pocket," in NDSS, 2014.

[4] Au, K. W. Y., et al., "PScout: Analyzing the Android permission specification," in Proc. ACM Conference on Computer and Communications Security, 2012.

[5] Cao, Y., et al., "Understanding users' privacy decisions in permission dialogs," ACM Transactions on Privacy and Security, 2021.

[6] Xiao, X., et al., "MPDroid: Detecting over-privileged apps using static analysis," IEEE Access, 2020.

[7] Alkinoon, M., et al., "A five-year study of Android permission evolution (2019–2023)," Computers & Security, 2025.

[8] Yilmaz, E., and Davis, J., "PRAM: Permission-based Android mobile privacy risk model," Journal of Information Security and Applications, 2023.

[9] Solanki, R., et al., "Evolution of Android permission model: A survey (2010–2022)," IEEE Access, 2026.

[10] Li, H., Sarathy, R., and Xu, H., "Understanding situational online information disclosure as a privacy calculus," Journal of Computer Information Systems, vol. 51, no. 1, pp. 62–71, 2010.

[11] Ren, J., Lindorfer, M., Dubois, D. J., Rao, A., Choffnes, D., and Vallina-Rodriguez, N., "Bug fixes, improvements, and privacy leaks: A longitudinal study of PII leaks across Android app versions," in Proc. Network and Distributed System Security Symposium (NDSS), 2018.

[12] Zimmeck, S., Wang, Z., Zou, L., Iyengar, A., Liu, B., Schaub, F., Wilson, S., Sadeh, N., and Bellovin, S. M., "Automated analysis of privacy requirements for mobile apps," in Proc. Network and Distributed System Security Symposium (NDSS), 2017.

[13] Wang, Y., Xiang, Y., and Zhou, W., "Android permission misuse detection based on machine learning," Future Generation Computer Systems, vol. 79, pp. 301–311, 2018.

[14] Tuncay, G. S., Demetriou, S., Ganju, K., Gunter, C. A., and Zhan, X., "Dr. Android and Mr. Hide: Fine-grained permissions in Android applications," in Proc. ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, 2016.

[15] Shuba, A., Leontiadis, I., Karagiannis, T., and Rodriguez, P., "A survey of mobile app permissions and user privacy risks," IEEE Communications Surveys & Tutorials, vol. 21, no. 3, pp. 2390–2412, 2019.