



Droid Scanner: App Security Management System

SE- 505: Software Project Lab - 2

Submitted by

Md. Shaikhul Islam (BSSE 1438)
Salsabila Zaman (BSSE 1443)

Supervised by

Prof. Md. Zulfiqar Hafiz
Institute of Information Technology
University of Dhaka

Supervisor's Approval: _____

Date: 09 March,2025

Acknowledgement

We would like to express our sincere gratitude to our supervisor, Prof. Md. Zulfiquar Hafiz Sir, for his invaluable guidance and continuous support throughout this project.

We also extend our thanks to Dr. Mohammad Shoyaib and Dr. Md. Shariful Islam Sir for their encouragement and for providing us with the knowledge that helped us develop Droid Scanner. Their insights and expertise have been instrumental in shaping our work.

Finally, we appreciate each other's dedication and teamwork in bringing this project to life.

Abstract

Droid Scanner is a security-focused desktop application designed to analyze Android APK files for potential threats. The system extracts permissions, intents from APK files and utilizes a machine learning model to classify apps as either benign or malicious. The tool offers multiple scanning options, including full device scans and single APK analysis, ensuring flexibility for users. To enhance usability, *Droid Scanner* provides a custom app management system, allowing users to whitelist or blacklist applications. The project was developed using PyQt5 for the interface, SQLite for database management, and Scikit-learn for machine learning. Key challenges included optimizing feature extraction, improving model accuracy, and ensuring efficient database storage.

Evaluation results demonstrate that *Droid Scanner* effectively detects malicious apps with high accuracy, making it a valuable tool for both general users and app developers. This report presents the system architecture, implementation process, and key findings, highlighting the effectiveness of the solution in improving Android security.

Table of Contents

Introduction	1
Background of the project	2
2.1 Malicious App Detection	2
2.2 APK Analysis Tools	2
2.3 SQLite Database	2
2.4 PyQt5 Framework	2
Project Overview	3
3.1 Key Features	3
3.2 Technical Stack	4
3.3. System Architecture	4
Comparison with Research-Based Approaches	5
4.1 Accuracy vs. Feature Usage	5
4.2 Output & Efficiency: Transparency and Performance	5
4.3 What sets Droid Scanner apart?	6
Description of the project	7
5.1 User Dashboard:	7
5.2 APK Upload and Analysis:	7
5.3 Scan Options:	7
5.4 Malicious App Detection:	7
5.5 Custom App Management:	7
5.6 Security Reports and Database Storage:	8
5.7 Developer Dashboard:	8
Data Based Modeling	9
6.1. Definition of Database Diagram	9
6.2. Final Data Objects:	9
6.3. Relationship between Objects:	10
6.4. Entity Relation (ER) Diagram:	10
6.5. Schema Diagram:	11
CLASS-BASED MODELING	12
7.1. CLASS BASED MODELING CONCEPT:	12
7.2. Class Cards	12

7.3. CRC Diagram:	16
User Manual	17
8.1. Portal Window:	17
8.2. Authentication Window:	17
8.2.1. Login Window:	17
8.2.2. Registration Window:	18
8.3. Main Window:	18
8.4. Scan Window:	19
8.5. Dashboard:	20
Challenges Faced	21
9.1 Extracting Features from APK Files	21
9.2 Training & Optimizing the Machine Learning Model	21
9.3 Making Large-Scale Scans Efficient	21
9.4 Ensuring Secure Data Storage & Privacy	21
9.5 Designing a Smart Whitelist & Blacklist System	21
9.6 Building a User-Friendly PyQt5 Interface	22
Future Scope and Conclusion	23
References	24

List of Figures

<i>Figure 1: ER Diagram</i>	<i>10</i>
<i>Figure 2: CRC Diagram</i>	<i>17</i>
<i>Figure 3. 1: Portal Window</i>	<i>17</i>
<i>Figure 3.2.1: Login Window</i>	<i>17</i>
<i>Figure 3. 3: Main Window.....</i>	<i>18</i>
<i>Figure 3. 4: Scan Window.....</i>	<i>19</i>
<i>Figure 3. 5: Dashboard.....</i>	<i>20</i>

Introduction

With the rise of mobile applications, security threats have become a growing concern for Android users. Malicious apps can steal personal data, track user activity, or even take control of a device without the user's knowledge. While security tools exist, many require technical expertise or fail to provide a clear, detailed analysis of an app's behavior. This leaves users uncertain about which apps to trust.

To bridge this gap, we developed *Droid Scanner*—a user-friendly desktop application that helps users analyze APK files and detect potential threats. The system extracts key features like permissions, intents from an app's metadata and uses a machine learning model to determine whether the app is safe or malicious. Users can choose between a full scan, which checks all installed apps on a connected device, or a quick scan for analyzing a specific APK file.

Beyond just scanning, *Droid Scanner* allows users to whitelist trusted apps and blacklist suspicious ones for ongoing monitoring. For Android developers, the tool serves as a security compliance assistant, helping them identify vulnerabilities in their applications before release.

This report walks through the motivation behind the project, the technical details of its implementation, and the challenges we faced along the way. We also present the results of our security analysis, showing how *Droid Scanner* can help make Android devices safer.

Background of the project

To build *DroidScanner*, we needed to dive into some key areas and tools to understand the best way to approach app security. Here's what we studied:

2.1 Malicious App Detection

Detecting malicious apps is a core part of our project. We researched various methods of spotting harmful apps, focusing on permissions, and intents. We needed to understand how malicious apps can threaten privacy or leak data. By analyzing APK files, we can spot these issues. We use machine learning to classify apps as either benign or malicious based on the behavior they exhibit, like which permissions they ask for or what actions they perform.

2.2 APK Analysis Tools

Analyzing APK files is crucial to understand what an app is doing behind the scenes. We looked into tools like **Apktool**, which helps us break down APK files, extract their AndroidManifest.xml, and check the permissions and intents the app requests. These extracted details are then compared with a database of known features, allowing us to spot any security risks or suspicious behavior.

2.3 SQLite Database

For storing all the data—like user information, scan results, and app features—we decided to go with SQLite. We needed a lightweight, efficient solution, especially since this is a desktop app. SQLite allows us to store and quickly retrieve data, so when we run scans on apps, we can easily compare their features to detect any malicious behavior. We also learned how to manage this data using SQL queries to make the process smooth and fast.

2.4 PyQt5 Framework

For the user interface, we chose PyQt5 because it's perfect for building desktop apps with Python. We researched how to use it to create a simple, intuitive design that makes it easy for users to upload APKs, run scans, and see the results. It also lets us integrate the backend (like the database and app analysis) seamlessly, ensuring that everything works together without a hitch.

Project Overview

3.1 Key Features

User Authentication & Dashboard:

- Secure login system with profile management.
- Users can update and edit their profiles.
- A centralized dashboard for easy access to features.

APK Analysis & Threat Detection:

- Users can upload APK files for security analysis.
- The system extracts permissions, intents from AndroidManifest.xml.
- Classifies apps as “Benign” or “Malicious” using a machine learning model.
- Scan results are securely stored in a User Database for future reference.

Multiple Scan Options:

- **Full Scan:** Scans all installed apps on the device.
- **Quick Scan:** Scans a single app on demand.

Custom App Management:

- Users can **whitelist** trusted apps to avoid repeated scans.
- Risky or suspicious apps can be **blacklisted** for continuous monitoring.

Developer-Focused Features:

- Dedicated **Developer Dashboard** to review scan reports.
- **Permissions Viewer** to analyze all requested permissions in an APK.
- Generates **detailed security reports** listing permissions, intents, and potential vulnerabilities.

3.2 Technical Stack

Component	Technology/Tool
Frontend	PyQt5, Qt Designer
Backend	Python 3.10
Database	SQLite (User & App Data)
Security	Secure authentication & access control
Machine Learning	Scikit-learn, Pandas, Joblib
APK Parsing	Apktool, ADB

3.3. System Architecture

APK Analysis & Detection Workflow

- 1. User Uploads APK File**
 - APK is extracted and analyzed for permissions and intents.
- 2. Feature Extraction & Classification**
 - Extracted features are matched against a predefined dataset.
 - The ML model classifies the app as **benign or malicious**.
- 3. Report Generation & User Access**
 - Users view detailed security reports in their dashboard.
 - Scan results are stored in the database for future reference.

Comparison with Research-Based Approaches

Several research models achieve high accuracy using deep learning and extensive feature extraction. Droid Scanner achieves similar accuracy (94%) with fewer, carefully selected features (135 key permissions & intents), making it more efficient and practical.

4.1 Accuracy vs. Feature Usage

Model	Accuracy (%)	Features Used
Droid Scanner	94%	210 permissions & intents
Metadata-Based ML	92.93%	2,137 permissions
FrameDroid	95-98%	API calls, permissions, bytecode, network flows
DetectBERT	96-97%	Smali code embeddings (DexBERT)
LensDroid	95-97%	Multi-view (API graphs, opcode, binary images)

4.2 Output & Efficiency: Transparency and Performance

Model	Output & Computational Requirements
Droid Scanner	Malware classification + extracted permissions, intents, manifest file + Runs on CPUs, lightweight SQLite database
Metadata-Based ML	Malware/benign label only + Batch processing of large metadata sets
FrameDroid	Model evaluation framework, no direct classification + Heavy ML model evaluations
DetectBERT	Black-box classification, no feature explanation + GPU-based deep learning model (DexBERT)
LensDroid	Visualized malware detection, harder to interpret + Deep learning, high computational cost

4.3 What sets Droid Scanner apart?

- Comparable accuracy with fewer features, reducing processing overhead.
 - Lightweight and fast, eliminating the need for GPUs.
- Transparent detection, providing clear insights into why an app is flagged.
- Real-time scanning, with future support for connected device analysis.

Droid Scanner is a practical, efficient alternative to deep-learning-based models, offering speed, transparency, and accessibility.

Description of the project

5.1 User Dashboard:

The application features a user-friendly dashboard that consolidates all functionalities for easy access. Users can update and edit their profiles, review their scan history, and manage app security settings.

5.2 APK Upload and Analysis:

Users can upload an APK file for security analysis. The system extracts features from the AndroidManifest.xml using Apktool, retrieving permissions and intents used by the application. These are then compared against a predefined database to detect any security risks.

5.3 Scan Options:

The system provides multiple scan options to offer flexibility:

- Full Scan – Analyzes all installed apps on a connected device.
- Quick Scan – Scans a specific app selected by the user.

These scans help identify malicious apps, ensuring better security for the user.

5.4 Malicious App Detection:

Once scanned, the system determines whether an app is benign or malicious based on its requested features. If an app is flagged as potentially harmful, users can review its details in the report.

5.5 Custom App Management:

To enhance user control, Custom App Management allows users to:

- Whitelist trusted apps to avoid redundant scans.
- Blacklist risky apps to ensure continuous monitoring.

This customization helps users tailor security preferences based on their needs.

5.6 Security Reports and Database Storage:

After each scan, the system generates a detailed security report outlining:

- App details
- Permissions used
- Intents
- Security classification
- Feature analysis

All scan results are securely stored in the SQLite database, allowing users to track past scans and reports for future reference.

5.7 Developer Dashboard:

For Android developers, *DroidScanner* functions as a compliance tool to help ensure app security before release. Developers can:

- Upload APKs for security analysis.
- View scan reports to assess potential vulnerabilities.

Data Based Modeling

6.1. Definition of Database Diagram

Data modeling is a process used to define and analyze data requirements needed to support the business processes within the scope of corresponding information systems in organizations. Therefore, the process of data modeling involves professional data modelers working closely with business stakeholders, as well as potential users of the information system.

Data Objects

A data object is a representation of composite information that must be understood by the software.

6.2. Final Data Objects:

1. **User**
 - Attributes: User_ID, Name, Role (Primary User/Developer), Password, Email, Verified, Verification_Token, Created_at
2. **App**
 - Attributes: App_ID, Package_Name, Version, Name, APK_File, Permissions, Intent, Status (Malicious/Benign)
 - Attributes
3. **Permissions_Intents**
 - Attributes: Feature_ID, Feature_Name
4. **Scan**
 - Attributes: Scan_ID, Type (Full/Quick), Date, App_ID (FK), User_ID (FK), Scan_Results
5. **Report**
 - Attributes: Report_ID, Scan_ID (FK), Report_File

6.3. Relationship between Objects:

Data Object	Relationship	Related Data Object
Scan	has	Report
User	performs	Scan
User	has	App
App	analyzed through	Scan
App	Has	Permissions_Intent

6.4. Entity Relation (ER) Diagram:

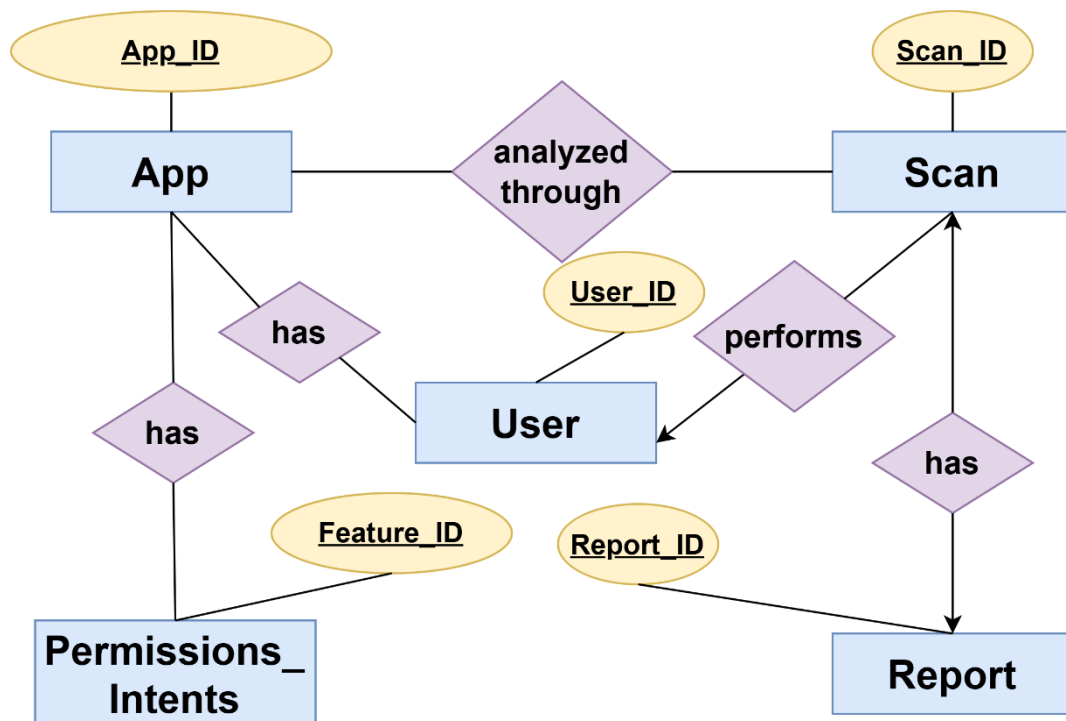


Figure 1: ER Diagram

6.5. Schema Diagram:

Data Object	Attribute	Type	Size
User	<u>User_ID</u> Name Password Email Verified Verification_Token Role Created_at	Integer Text Text Text Integer Text Text Timestamp	100 100 50 100 100 100 50 100
App	<u>App_ID</u> Package_Name Name Status APK_File	Integer Varchar Varchar Blob Text	100 100 50
Permissions_Intent	<u>Feature_ID</u> Feature_Name	Integer Varchar	100
App_features	<u>App_Feature_ID</u> App_ID (FK) Feature_ID (FK)	Integer Integer Integer	
Scan	<u>Scan_ID</u> Type App_ID (FK) User_ID (FK) Scan_Results Date	Integer Varchar Varchar Varchar Text DateTime	100 50 100 100
Report	<u>Report_ID</u> Scan_ID (FK) Report_File	Integer Integer Blob	100 100

CLASS-BASED MODELING

7.1. CLASS BASED MODELING CONCEPT:

Class-based modelling represents the objects that the system will manipulate, the operations that will be applied to the objects, relationships between the objects and the collaborations that occur between the classes that are defined.

Class-based modelling represents the objects that the system will manipulate, the operations that will be applied to the objects, relationships between the objects and the collaborations that occur between the classes that are defined.

Here the list of Final Classes:

- I. Authentication
- II. AuthWindow
- III. MainWindow
- IV. Scan
- V. Report
- VI. ML Model
- VII. Dashboard
- VIII. Database

7.2. Class Cards

After identifying our final classes we have generated the following class cards.

Authentication	
Attributes	Methods
	validateUser()
	verifyEmail()
	generateOTP()
	createJWT()
Responsibilities	Collaborators
Authentication of users	Database
Send and verify OTP	Email_service

AuthWindow	
Attributes	Methods
	validate_user()
	show_verification_dialogue()
	open_main_window()
Responsibilities	Collaborators
Manage login and registration	Authentication
Connect to App interface	MainWindow

MainWindow	
Attributes	Methods
u_id	choose_file()
user_name	open_scan_window()
result	open_dashboard()
pdf_data	update_scan_result()
	get_user_credentials()
	train_model()
	get_user_name()
	show_report()
Responsibilities	Collaborators
Manage user accounts and profiles.	Dashboard
Scanning initiate	Scan
Train model	ML model
Upload apk from PC	Dashboard
Download report	Report

Scan	
Attributes	Methods
Scan_ID	check_devices()
u_id	list_installed_packages()
packages	handle_quick_scan()
	handle_full_scan()
	extract_apk()
	extract_features()
	extract_manifest()
	classify_apk()
	status_update()
	generate_report()
Responsibilities	Collaborators
Analyze apps and generate detailed reports.	ML Model, Report
Store App information and report	Database
Show report	MainWindow

Report	
Attributes	Methods
Report_ID	generate_Report_content()
pdf_data	save_report_to_database()
	generate_report()
	download_report()
Responsibilities	Collaborators
Get Scan Details	Scan
Provide Scan Report	Dashboard

ML Model	
Attributes	Methods
csv_file	trainModel()
classifier	
accuracy	
confusion_matrix	
Responsibilities	Collaborators
Analyze extracted APK data and classify apps as malicious or benign.	Scan

Dashboard	
Attributes	Methods
u_id	show_credential()
	show_history()
	show_applist()
	load_stored_data()
	update_user_in_database()
	show_applist()
	load_features()
	add_to_list()
	remove_from_list()
Responsibilities	Collaborators
Display and update scan history	Database, Scan
Update and Show app lists	Database

Database	
Attributes	Methods
	add_user()
	check_login()
	get_verification_token()
	store_apk()
	update_user_credential()
	log_scan()
Responsibilities	Collaborators
Send app permissions and intents.	MLModel, Scan
Store and manage user credentials	AuthWindow

7.3. CRC Diagram:

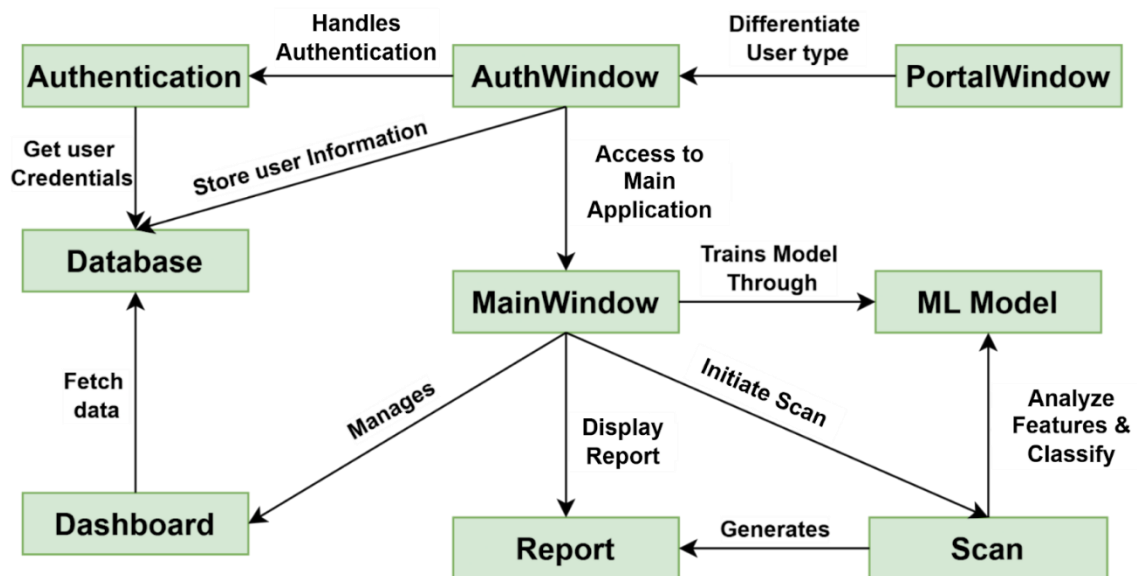


Figure 2: CRC Diagram

User Manual

8.1. Portal Window:

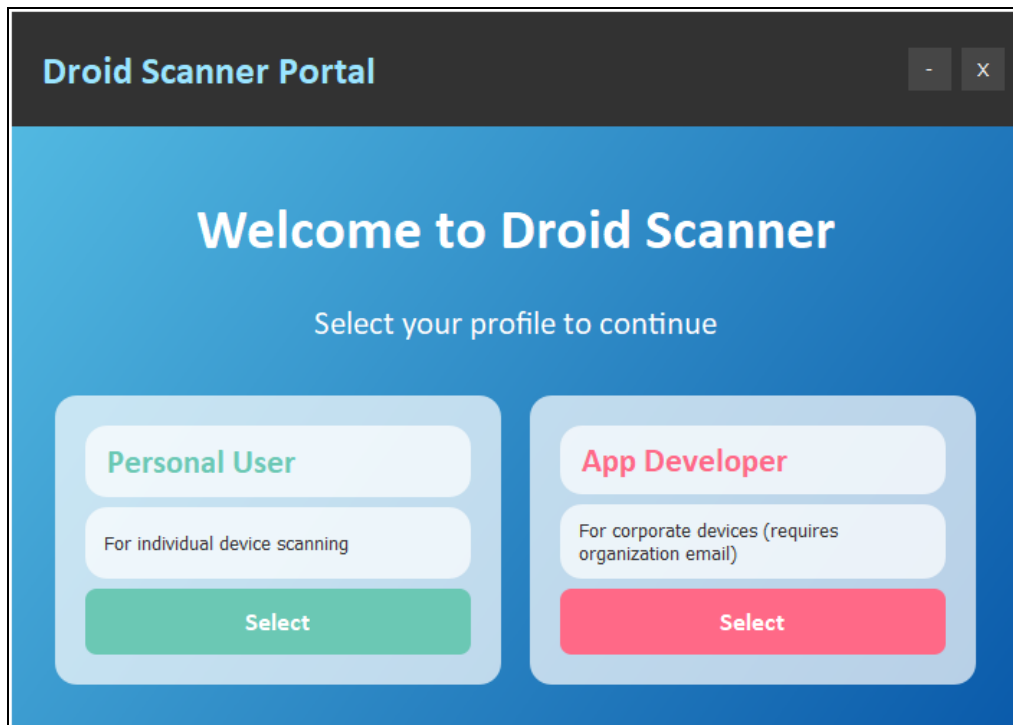


Figure 3. 1: Portal Window

8.2. Authentication Window:

8.2.1. Login Window:

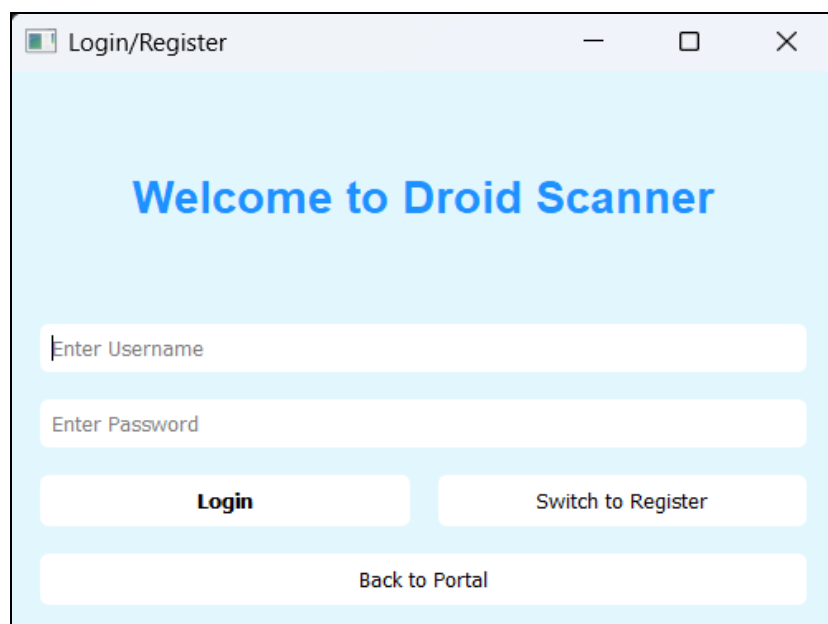
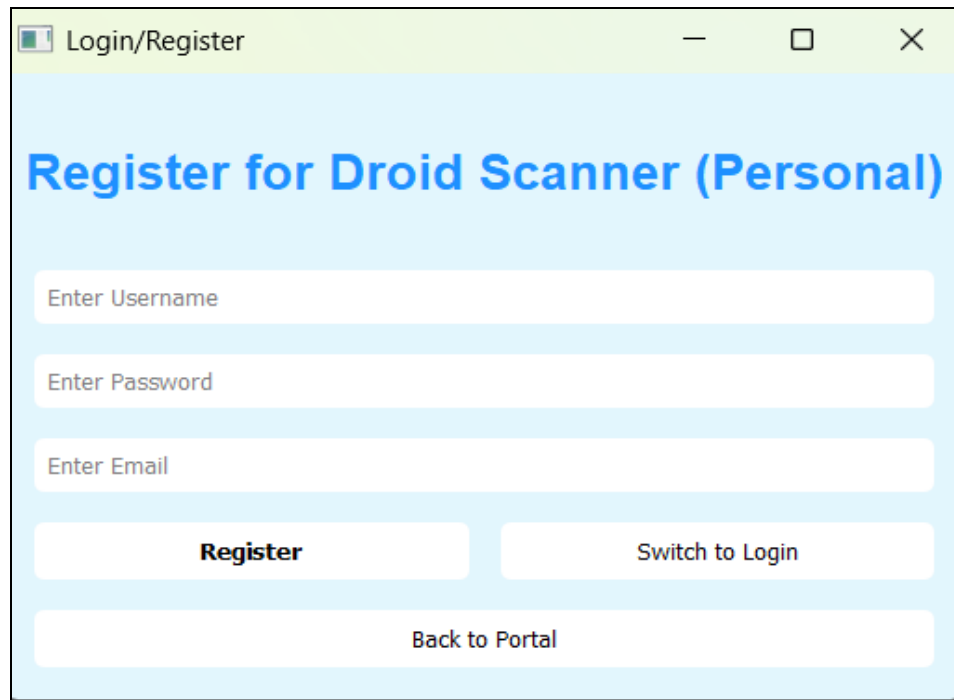


Figure 3.2.1: Login Window

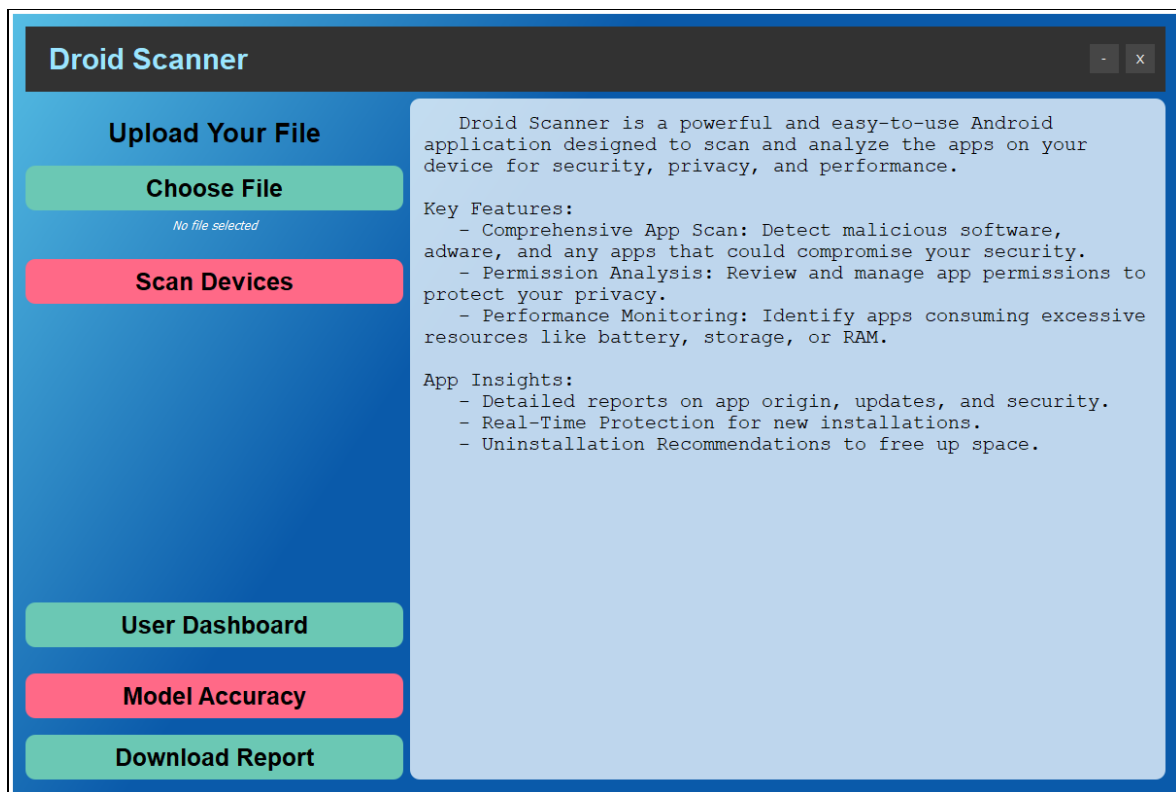
8.2.2. Registration Window:



The registration window is titled "Login/Register" and "Register for Droid Scanner (Personal)". It contains three input fields: "Enter Username", "Enter Password", and "Enter Email". Below these fields are two buttons: "Register" and "Switch to Login". At the bottom, there is a button labeled "Back to Portal".

Figure 3.2.2: Registration Window

8.3. Main Window:



The main window is titled "Droid Scanner". It features a sidebar on the left with the following options: "Upload Your File", "Choose File" (with "No file selected" below it), "Scan Devices", "User Dashboard", "Model Accuracy", and "Download Report". The main content area on the right contains the following text:

Droid Scanner is a powerful and easy-to-use Android application designed to scan and analyze the apps on your device for security, privacy, and performance.

Key Features:

- Comprehensive App Scan: Detect malicious software, adware, and any apps that could compromise your security.
- Permission Analysis: Review and manage app permissions to protect your privacy.
- Performance Monitoring: Identify apps consuming excessive resources like battery, storage, or RAM.

App Insights:

- Detailed reports on app origin, updates, and security.
- Real-Time Protection for new installations.
- Uninstallation Recommendations to free up space.

Figure 3. 3: Main Window

8.4. Scan Window:

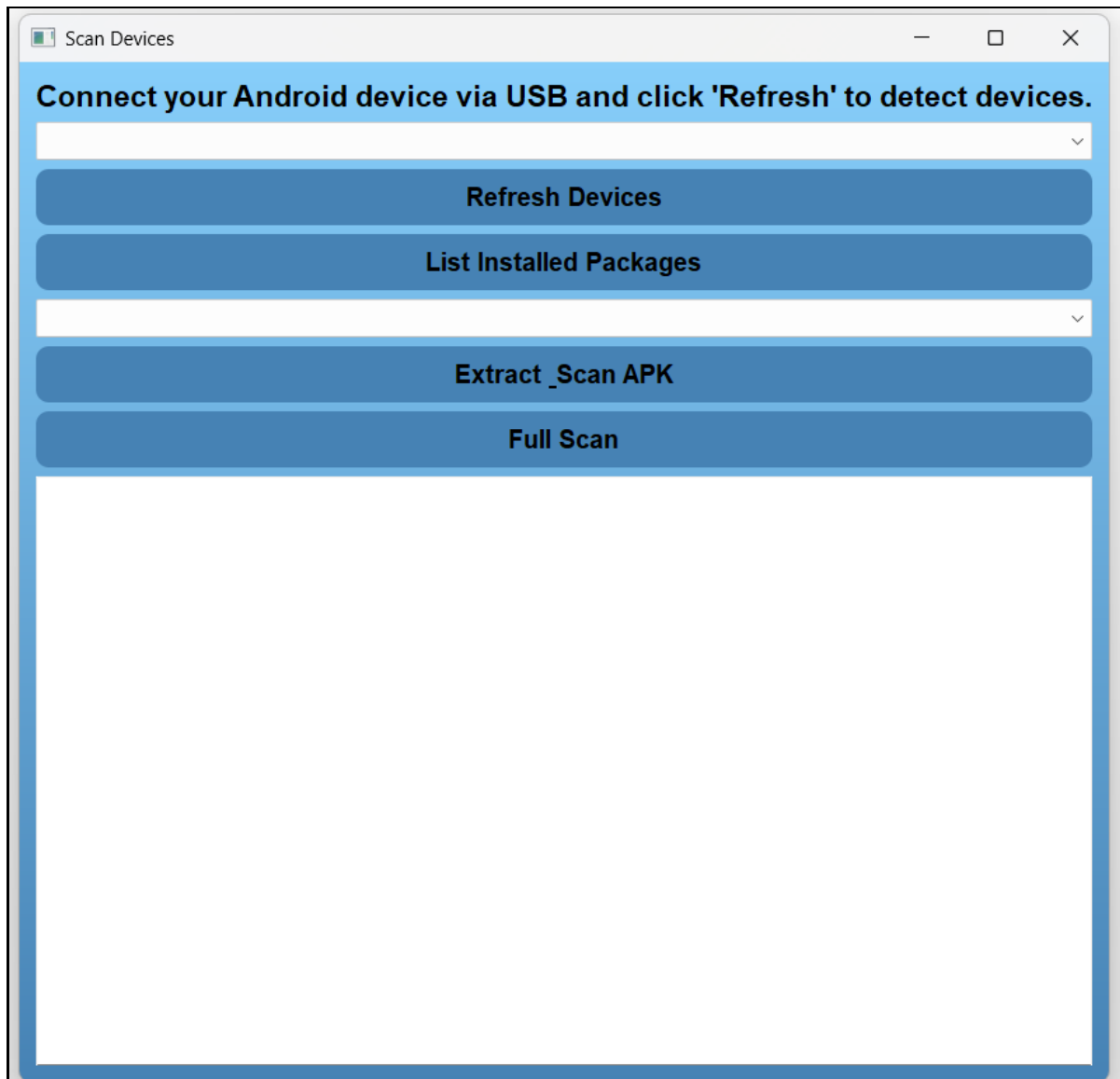


Figure 3. 4: Scan Window

To connect your device to our application, follow these steps:

1. **Enable USB Debugging** – On your Android device, navigate to *Settings > About phone* and tap Build Number seven times to unlock Developer Options. Then, go to *Settings > Developer Options* and enable USB Debugging.
2. **Connect Your Device** – Use a USB cable to connect your device to the computer. Ensure that the connection mode is set to File Transfer (MTP), PTP, or MIDI, as the "Charge Only" mode will prevent detection.
3. **Device Detection** – Once connected, the application will automatically detect your device and proceed with the scanning process.

8.5. Dashboard:

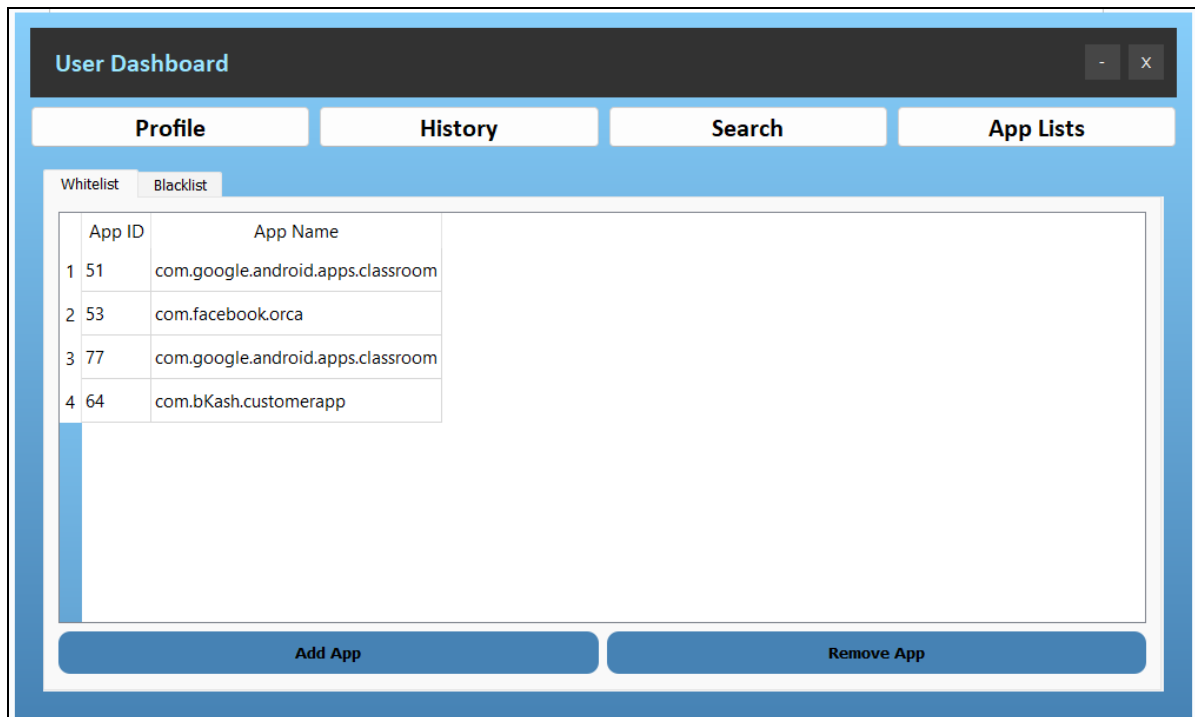


Figure 3. 5: Dashboard

Challenges Faced

Building *Droid Scanner* wasn't without its hurdles. Along the way, we encountered several challenges that pushed us to refine our approach, rethink our strategies, and improve our system. Here's a look at some of the key obstacles we tackled and how we overcame them:

9.1 Extracting Features from APK Files

Extracting permissions, intents from APK files was more complicated than we initially expected. While tools like Apktool and ADB gave us raw data, making sense of that data—structuring it in a way that was meaningful and accurate—was a challenge. We had to ensure that no critical security details were overlooked while also avoiding redundant information. A lot of trial and error went into fine-tuning this process to get the most reliable results.

9.2 Training & Optimizing the Machine Learning Model

Getting our ML model to correctly classify apps as benign or malicious was a major challenge. Finding the right dataset, dealing with imbalanced data, and selecting the best algorithm all required multiple rounds of testing. We had to strike a balance between accuracy and performance—avoiding overfitting while also reducing false positives that could wrongly flag safe apps. Fine-tuning the model to improve reliability was a time-consuming but essential step.

9.3 Making Large-Scale Scans Efficient

Running a Full Scan—which analyzes all installed apps on a device—proved to be a performance challenge. Processing multiple apps at once required significant computing power, and we had to ensure that the system wouldn't slow down during scanning. To optimize performance, we implemented efficient database queries and parallel processing techniques, making the scans faster and more seamless for users.

9.4 Ensuring Secure Data Storage & Privacy

Since we store scan results, data security and privacy were top priorities. We needed a database structure that kept user data protected while still allowing easy access to scan histories. More importantly, we had to make sure no personally identifiable information (PII) was stored in a way that could compromise security. A lot of careful planning went into designing a system that strikes the right balance between accessibility and protection.

9.5 Designing a Smart Whitelist & Blacklist System

Giving users the ability to whitelist trusted apps and blacklist risky ones seemed simple at first, but in reality, it required thoughtful design. We needed to ensure that users could easily

manage their lists without mistakenly marking an app as safe or dangerous. The UI had to be intuitive, and the backend had to support real-time updates without errors. It took several refinements to get it working smoothly.

9.6 Building a User-Friendly PyQt5 Interface

Creating an interactive desktop application using PyQt5 brought its own set of challenges. We wanted the UI to be intuitive and responsive, making it easy for both regular users and developers to navigate. Achieving this meant refining the design multiple times, improving responsiveness, and ensuring that every feature felt natural to use.

Future Scope and Conclusion

Looking ahead, there's a lot of potential to expand *Droid Scanner* by improving how it detects malicious behaviors. One way to do this is by adding more features and APIs to analyze app behaviors and network activities in greater detail, which would make the classification process even more accurate. Additionally, by exploring newer machine learning models and techniques, we can improve the app's ability to spot threats more effectively. It would also be great to introduce real-time scanning, so users could get instant alerts about any potential risks while installing or updating apps. Keeping the app's database up-to-date with newly discovered features, permissions, and intents will be crucial as the Android ecosystem keeps evolving.

To improve the app, we can:

- **Broaden the behavior analysis** by looking into app network requests and how they interact with device hardware.
- **Expand the dataset** to include more permissions, intents, and behaviors.
- **Introduce real-time scanning** to notify users of potential threats during installation or app updates.

In conclusion, *Droid Scanner* is a valuable tool for protecting mobile devices by analyzing APK files and classifying them as either malicious or benign. It offers an intuitive interface and relies on an efficient machine learning model to provide reliable predictions. Our goal is for *Droid Scanner* to continue helping mobile developers and users maintain high security standards. As the landscape of mobile threats changes, we hope to keep improving the app to provide even better protection in the future.

References

- [1] J. Liu, J. Zeng, F. Pierazzi, L. Cavallaro, and Z. Liang, “Unraveling the Key of Machine Learning Solutions for Android Malware Detection,” *arXiv preprint*, 2024. [Online].
- [2] T. Sun, N. Daoudi, K. Kim, K. Allix, T. F. Bissyandé, and J. Klein, “DetectBERT – A New Approach to Android Malware Detection,” *arXiv preprint*, 2024. [Online].
- [3] Z. Meng, J. Zhang, J. Guo, W. Wang, W. Huang, J. Cui, H. Zhong, and Y. Xiong, “LensDroid – Visualizing Android App Behaviors for Malware Detection,” *arXiv preprint*, 2024. [Online].
- [4] A. Hefter, C. Sendner, and A. Dmitrienko, “Metadata-Based Malware Detection on Android Using Machine Learning,” *arXiv preprint*, 2023. [Online].