Patronus:

AI-Powered Anti-Malware and Application-Level Network Threat Detection System for Android Devices

---

Submitted by:

Aml Ibrahim Kassem

Nouran Abdelsalam Mohamed

John George Farid

Ranwah Gamal Asala

Menna Allah Ahmed Saad

Suhaila Adel Ali

Supervised by:

Prof. Dr Yasser Fouad

Graduation Project II Documentation 2025

Faculty of Computers and Data Science - Alexandria University

Cyber Security Department

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1. Problem Statement

As mobile devices become essential to everyday life, they are increasingly targeted by sophisticated cyber threats. Users rely on smartphones for communication, banking, file sharing, and accessing sensitive data—yet most are unaware of the risks they face from malicious apps, insecure public Wi-Fi networks, and hidden background connections to harmful servers. Existing mobile security solutions often focus narrowly on detecting known malware and lack intelligent, adaptable defenses against modern threats, especially zero-day attacks and network-based exploits.

One major vulnerability is that users frequently install new apps without knowing whether they are safe. Many of these apps may silently perform malicious activities in the background, such as data theft or unauthorized communication with malicious IP addresses. Another common issue is the use of public Wi-Fi networks, which may be open, unencrypted, or even maliciously set up to intercept user data. Traditional antivirus tools rarely provide users with real-time insights into the safety of the networks they connect to or the behavior of apps after installation.

This project addresses these growing challenges by developing an advanced Android security application that offers comprehensive protection at both the device and network levels. The application monitors newly installed and existing apps **based on the user-selected security mode**. When monitoring is active, apps are analyzed for suspicious behavior patterns using AI-powered, behavior-based detection methods. If malicious behavior is detected, the system performs automated threat mediation, such as alerting the user, restricting the app's access, or suggesting its removal.

To improve user experience and understanding, the application integrates a **chatbot powered by a large language model (LLM)**. This intelligent assistant helps users understand security alerts, provides real-time advice, explains technical details, and improves overall interaction with the system.

1.2. Objective

The objective of this project is to develop an Android application that:

- Implement **AI-based malware detection** to monitor newly installed and existing apps, analyzing their behavior to identify potential security threats in real-time.
- Enable three configurable security modes (**Low, Balanced, High**) that allow users to customize the level of monitoring and protection according to their performance and security preferences.
- Incorporate an **App-Level Network Monitor** that continuously checks app connections against a dynamically updated database of known malicious IP addresses, providing proactive defense against network-based attacks.
- Integrate a **Wi-Fi Network Scanner** that evaluates the security of public Wi-Fi networks by analyzing key parameters such as network type, security status, and speed, while offering troubleshooting recommendations.
- Offers a user-friendly interface for monitoring device security and responding to potential threats with minimal impact on device performance and battery life.
- Embed an **AI-driven chatbot** that interacts with users, providing immediate explanations of security alerts, answering questions, and offering tailored recommendations to enhance user awareness and control over their device's security.

1.3. Contribution

This project makes significant contributions to the field of mobile security by addressing the growing concerns associated with malicious apps and insecure public Wi-Fi networks. The key contributions of this project are:

- This project contributes to mobile security by developing an Android application that:
- Utilizes AI-based malware detection to identify threats in real-time, including zero-day attacks, without relying on traditional signature-based methods.
- Offers three customizable security modes (Low, Balanced, High) that allow users to adjust the level of protection and performance according to their needs.
- Includes an App-Level Network Monitor that detects connections to malicious IP addresses, adding a layer of defense against network-based attacks.
- Features a Wi-Fi network scanner to assess the safety of public Wi-Fi connections, helping users avoid insecure networks.

- Integrates an AI-driven chatbot for immediate user support and guidance on security issues.

1.4. Organization

List of Figures and Tables: Provide references to all visual elements, including diagrams, charts, and tables used throughout the thesis.
Chapter 1: Introduces the project by outlining the problem statement, research objectives, and key contributions. It also provides an overview of how the thesis is organized.
Chapter 2: Establishes the conceptual framework by covering:

- The Android Security Model and Permission System
- Common types of Android malware attacks
- Related academic and industry work
- A detailed comparative analysis of similar mobile security applications

Chapter 3: Describes the core methodologies applied in the project, including:

- An overview of the datasets used (DREBIN, TUANDROMD, COLCOM)
- Feature engineering techniques, such as malware permission profiling
- The neural network model architecture
- Evaluation strategies and generalization testing
- The integration of a Large Language Model (LLM) component

Chapter 4: Explains the design and development of the *Patronus* application, including:

- Component architecture and detailed module workflows
- Real-time system monitoring, network diagnostics, and AI-powered malware scanning
- The use case diagram and user interface design, supported by screenshots of each major screen

Chapter 5: Summarizes the project findings, discusses the insights gained during development, and outlines future improvements such as support for Android 10+ and voice interaction in HelpBot.
Appendix: Contains relevant code snippets and technical details to support implementation and reproducibility.
References: Lists all cited academic sources, tools, datasets, and other materials referenced in the thesis.

Chapter 2

Background and Similar Software

2.1. Introduction

This chapter establishes the foundational knowledge necessary to understand the security landscape of Android systems and the relevance of mobile security applications. It begins by introducing the conceptual framework that includes an overview of the Android security model, permission system, and various types of malware attacks that pose significant threats to Android devices. It then presents a comprehensive review of related academic work and analyzes the capabilities of popular mobile security software. This comparative analysis highlights strengths and limitations of existing solutions and motivates the need for developing improved methodologies in Android malware detection.

2.2. Conceptual Framework

2.2.1. Android Security Model Overview

The Android security model relies on app sandboxing to isolate apps from one another, ensuring that one app cannot access another app's data without explicit authorization. Each app is assigned a unique UID (User ID) and operates within a dedicated process and data directory, preventing unauthorized access to its files. This isolation occurs both at the process and file levels, safeguarding data privacy and integrity. To allow apps to interact with system resources such as hardware, Wi-Fi, or the camera, Android uses permissions, which grant specific access rights for resource sharing while maintaining the security of app boundaries. This approach restricts app interaction unless explicitly authorized by the user, enhancing the overall security of the Android system.

2.2.2. Android Permission Model

Android app permissions are designed to protect users' privacy by limiting access to restricted data and actions. Examples of restricted data include system state information and users' contact details, while restricted actions encompass activities like connecting to a paired device or recording audio. If an app needs access to restricted data or actions to function properly, it's recommended to follow these steps:

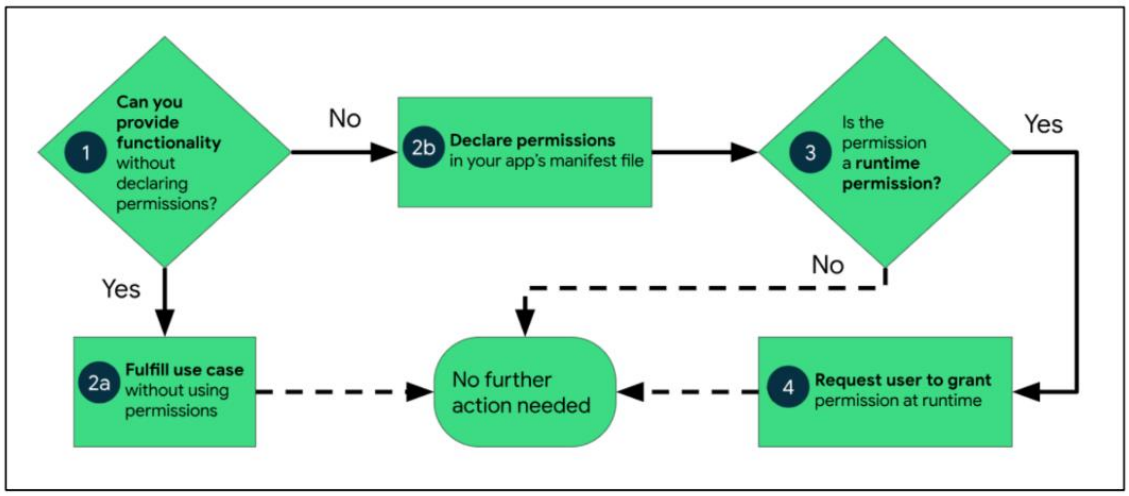

Figure 2.1 Workflow for using permissions on Android (Source: Android Developers Documentation)

1. Permissions

Android app permissions are designed to safeguard user privacy by restricting access to sensitive data and actions. For example, restricted data includes system state information and user contacts, while restricted actions involve activities such as connecting to a paired device or recording audio.

Android permissions are categorized into different types, each specifying the scope of restricted data or actions an app can access. Each permission type is also assigned a corresponding protection level. The types are as follows:

1.1. Install-time Permissions

Install-time permissions provide limited access to restricted data or allow minor actions that don't significantly impact the system or other apps. These permissions are presented to the user on the app store's details page, and they are automatically granted when the app is installed.

1.1.1. Normal Permissions

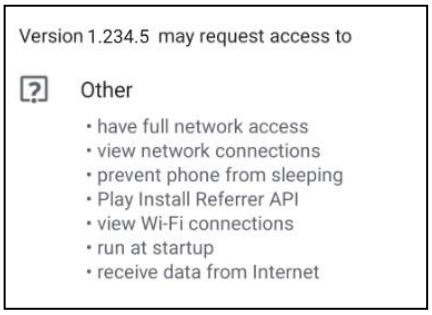

Figure 2.2 Install-Time Permissions on an App Store

These permissions grant access to data and actions outside an app's sandbox, but they pose minimal risk to user privacy or the functioning of other apps. The system assigns these permissions a normal protection level.

1.1.2. Signature permissions

These permissions are granted only to apps signed with the same certificate as the app or the operating system (OS) that defines the permission. This ensures that only trusted apps, signed with the same certificate, can access certain privileged actions or data. The system assigns the signature protection level to signature permissions.

1.2. Runtime Permissions

Runtime permissions, or dangerous permissions, provide access to restricted data or allow actions that can significantly impact the system or other apps. These permissions must be explicitly requested by the app at runtime before they can be used to access such data or perform these actions. When an app requests a runtime permission, the system presents a runtime permission prompt. The system assigns the dangerous protection level to runtime permissions.

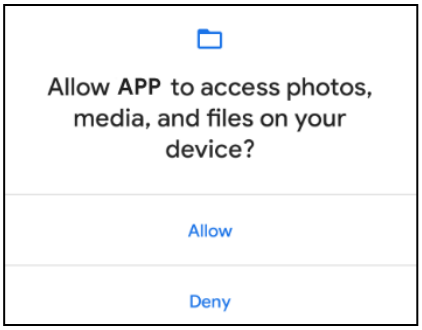

Figure 2.3 Runtime Permission Prompt

1.3. Special Permissions

Special permissions are defined by the platform and OEMs to control access to specific, sensitive app operations that could impact system security or functionality. These permissions are usually granted for powerful actions, such as drawing over other apps or accessing system resources, which require additional protection to prevent misuse. Only the platform and OEMs can define these permissions, ensuring that critical operations are properly restricted. The system assigns the appop protection level to special permissions.

1.4. Permission Groups

Permissions are categorized into groups based on related functions, like SMS permissions for sending and receiving messages. This grouping reduces the number of prompts users see, as permissions within the same group are requested together in one prompt.

2.2.3. Types of Malware attacks

2.2.3.1. Adware

Adware is a type of software designed to display unwanted advertisements on a user's device. It often operates within apps or while browsing the internet. Adware can vary in its impact, with some being relatively harmless by simply generating revenue through ads, while others can be malicious, tracking personal data and degrading system performance.

There are two main types of adware: non-malicious and malicious. Non-malicious adware primarily focuses on showing ads within apps or websites for revenue purposes, usually using well-known advertising platforms like Google AdMob. Malicious adware, however, goes beyond simple ad-serving by tracking personal data, showing intrusive ads such as overlay ads, and potentially stealing sensitive information from the device.

Adware typically works through a specific workflow on Android devices. It is often bundled with other apps, especially those from unofficial sources, and may request unnecessary permissions during installation, such as access to the user's contacts, location, and camera. Once installed, adware begins displaying ads within apps, as banners, pop-ups, or full-screen overlays, and sometimes sends push notifications that display ads even when the app is not in use. Additionally, adware tracks user behavior, such as the apps used and websites visited, and collects personal data like contacts or messages, which it sends to third-party advertisers.

Adware significantly impacts the Android system by slowing down the device due to its resource consumption, such as CPU and memory, as it continuously loads ads. This leads to battery drain from constant background activity and increased data usage as the device frequently connects to ad servers to fetch new ads or send collected data. The user experience can be disrupted by

intrusive ads appearing outside their intended context, and malicious adware can pose privacy risks by collecting and transmitting personal information without the user's consent.

Permissions typically requested by adware include internet access to serve ads and track user behavior, permission to display ads over other apps, permission to run at startup for persistent ad-serving, and access to location, contacts, or camera to gather personal data for targeted advertising. Detecting adware involves looking for signs such as unexpected pop-up ads, unfamiliar apps on the device, increased data usage, faster battery drain, decreased device performance, and apps that request excessive permissions not aligned with their intended functionality, like a simple flashlight app requesting access to contacts.

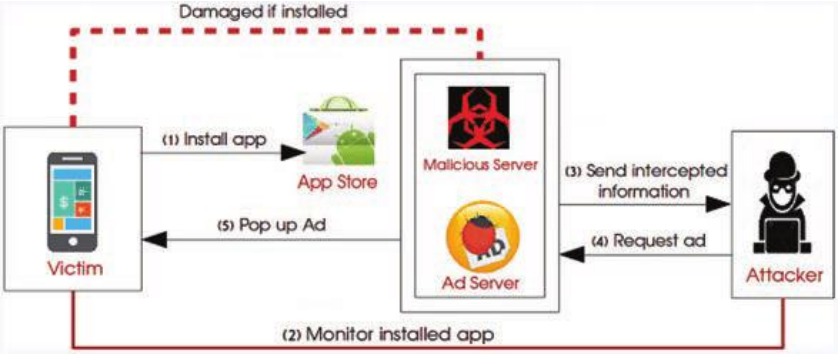

Figure 2.4 Adware Workflow

2.2.3.2. Android ransomware

Android ransomware is a form of malware that infects Android devices, locking users out of their data or encrypting files and demanding a ransom to restore access. It commonly appears in two forms: screen-locking ransomware, which blocks access to the device's screen with a ransom message, and file-encrypting ransomware, which encrypts critical files and demands payment for a decryption key. Notable examples include Simplelocker and Lockerpin.

There are two main types of ransomware: Screen-locking ransomware prevents access to the device by locking the screen and displaying a ransom note, demanding payment to unlock the device. File-encrypting ransomware, on the other hand, encrypts user data, making it unusable until the victim pays the ransom to obtain a decryption key that restores access to their files.

The workflow of Android ransomware begins with infection. Attackers embed malicious code in seemingly legitimate apps, often targeting popular applications to increase the likelihood of installation. Once installed, the ransomware initiates its malicious actions. During installation, the ransomware app requests sensitive permissions, such as administrative rights, which allow it to lock the device or encrypt files. After gaining the necessary permissions, the ransomware app connects to a Command & Control (C&C) server, where it awaits instructions from the attacker, such as transferring user data or initiating file encryption. To evade detection, ransomware often

disables or kills background anti-malware processes, allowing it to execute its malicious payload undetected.

Once activated, the ransomware carries out its malicious task. Depending on the attacker's instructions, the app either locks the device or encrypts user data using encryption algorithms like AES. Afterward, a ransom note is displayed on the device, either as a pop-up or by changing the wallpaper, demanding payment—usually in cryptocurrency—in exchange for the decryption key. However, even after paying the ransom, there is no guarantee that the attacker will provide the correct key, leaving victims at risk of permanent data loss.

Ransomware has significant effects on the Android system. It can lock the device entirely, preventing the user from accessing it. It encrypts important data, making it inaccessible without the correct decryption key. The ransomware's background activities, such as encryption and data transfer, consume considerable system resources, which can lead to performance issues. Additionally, there is a risk of data loss, as even paying the ransom does not always result in data recovery. Security and privacy risks are also prevalent, as the ransomware may transfer sensitive user data, such as contacts and documents, to the attacker.

To carry out these actions, ransomware apps often request various permissions. BIND_DEVICE_ADMIN grants administrative access, enabling the app to lock the device or wipe data. KILL_BACKGROUND_PROCESSES allows the ransomware to stop anti-malware or security apps running in the background. WRITE_EXTERNAL_STORAGE enables the modification or encryption of files on external storage. INTERNET access is required to connect to C&C servers, while RECEIVE_BOOT_COMPLETED ensures that the ransomware persists by restarting after the device reboots. Lastly, READ_CONTACTS allows the ransomware to access contact data, which can be exploited to further spread the malware.

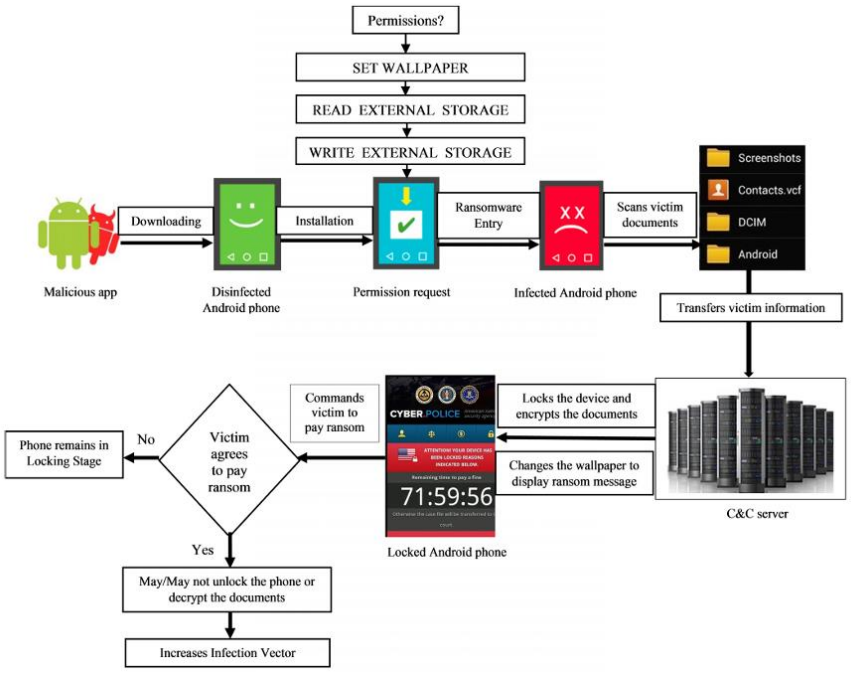

Figure 2.5 Ransomware Workflow

2.2.3.3. Trojan

A Trojan, or Trojan horse, is a type of malicious software designed to deceive users by pretending to be legitimate or desirable applications, such as games, system updates, or utilities. Unlike traditional viruses, Trojans do not self-replicate but instead rely on user activation to cause harm. Once installed and activated, Trojans can perform a range of malicious activities, including stealing sensitive information, sending unauthorized premium SMS messages, or turning devices into botnet components for larger-scale cyberattacks. This makes Trojans a particularly insidious threat to Android devices.

Trojans come in various forms, each targeting specific vulnerabilities on Android devices. Trojan/Fakeinst impersonates legitimate apps to trick users into installation, covertly executing unauthorized actions. Trojan/FakeToken targets SMS-based two-factor authentication (2FA) by intercepting and collecting token codes to bypass security mechanisms. Trojan/MarketPay.A facilitates unauthorized purchases in app marketplaces, leading to unexpected charges for users. Another type, Trojan-Proxy/NotCompatible.A, turns infected devices into bots that hackers use for network-based attacks like Distributed Denial of Service (DDoS). Finally, Trojan/AdSMS.A sends premium-rate SMS messages without user knowledge, modifies network settings, and steals sensitive device data such as IMEI numbers and phone details.

A Trojan on an Android device typically follows a structured workflow to infiltrate and exploit the system. Initially, the Trojan disguises itself as a legitimate app, such as a game or system update,

to gain the user's trust. It is often downloaded from unofficial sources, malicious links, or repackaged versions of popular apps. During installation, the Trojan requests excessive permissions, such as access to SMS, Wi-Fi settings, phone identity, and storage, often using social engineering tactics to make these requests appear necessary. Once installed, the Trojan executes its malicious activities, which can include stealing sensitive data like phone numbers, IMEI numbers, and two-factor authentication codes, as well as sending unauthorized premium-rate SMS messages or altering system settings. Trojans employ stealth tactics to avoid detection, such as hiding their app icons or using misleading names like "andiord.system.providers," while silently performing activities like sending premium SMS messages or changing Wi-Fi settings. They may continue to run in the background, silently performing harmful actions like connecting to remote servers for further instructions or downloading additional malicious payloads. Some Trojans even turn infected devices into bots, contributing to large-scale cyberattacks such as Distributed Denial of Service (DDoS) attacks. Ultimately, the Trojan compromises the device's security, degrades its performance, and can lead to financial loss or further exploitation of the user's personal information.

Trojans can severely compromise Android devices by stealing sensitive information, gaining unauthorized control, and turning the device into a tool for cyberattacks. They can collect critical data such as IMEI numbers, phone numbers, and network details, which can be used for identity theft or sold on the black market. In addition to data theft, Trojans can modify system settings, including Wi-Fi configurations, and perform unauthorized actions like sending premium-rate SMS messages, leading to unexpected financial charges for users. More sophisticated Trojans may integrate infected devices into botnets, which are used for large-scale cyberattacks such as Distributed Denial of Service (DDoS) attacks. Additionally, Trojans can cause significant resource drain, leading to high CPU and battery consumption that degrades device performance.

Permissions Needed from the Android System To execute their malicious activities, Trojans often request critical permissions that exceed the requirements of legitimate apps. These permissions include reading SMS messages to intercept and steal sensitive information, such as two-factor authentication codes, and sending SMS messages to send premium-rate messages without user consent. Trojans also request permissions to modify Wi-Fi connection settings, allowing them to alter network configurations for further exploitation. Additionally, they often demand access to phone identity details, such as IMEI numbers and phone numbers, which are sent to remote servers for malicious use. Other permissions may include access to storage to analyze and manipulate files or the ability to install packages, enabling the Trojan to download and install additional malicious payloads without user consent.

2.2.3.4. Spyware

Spyware is a malicious software type designed to infiltrate devices, collect sensitive information, and transmit it to unauthorized third parties without the user's consent. On Android devices, spyware poses significant threats by compromising privacy, tracking user activities, and stealing personal data such as login credentials, financial information, browsing history, and call logs. The collected data is often sold or exploited for malicious purposes, making spyware a critical concern for Android users.

Spyware can be classified into several types based on its functionality and purpose. Infostealers collect specific data such as login credentials and private conversations, while keyloggers record every keystroke made on an infected device, capturing sensitive information. Rootkits provide attackers with undetectable access to the system, exploiting vulnerabilities and making them difficult to eliminate. Red Shell spyware, often bundled with software, tracks user activity for marketing purposes but can pose privacy risks. Tracking cookies monitor browsing activity for advertising, and system monitors capture various user interactions across the device. Mobile spyware specifically targets smartphones, accessing sensitive data like messages, call logs, contacts, browsing history, and GPS locations. Some types can even remotely activate the microphone or camera.

The workflow of spyware on Android devices typically involves four stages. First, spyware infiltrates the device through malicious websites, infected files, unauthorized app downloads, or phishing schemes. Once installed, it begins monitoring user activities, such as keystrokes, browsing behavior, and app usage. In the next stage, spyware collects sensitive information, including login credentials, GPS data, and communication logs. Finally, the collected data is transmitted to remote servers or sold on illicit platforms, enabling activities like identity theft and fraud.

Spyware significantly impacts Android devices, leading to data theft, privacy violations, and financial losses. The stolen data is often used for identity theft, granting attackers unauthorized access to user accounts or financial resources. Additionally, spyware strains system resources, causing slower operation, high battery consumption, and even system crashes. It can also disrupt browsing experiences by redirecting users to unwanted websites, altering default search engines, and displaying excessive pop-ups, ultimately compromising the device’s functionality and user experience.

Spyware exploits specific Android permissions to perform its malicious activities. Application monitoring permissions, such as PACKAGE_USAGE_STATS and QUERY_ALL_PACKAGES, enable tracking of app usage patterns and identification of installed applications. Network monitoring permissions, including INTERNET, ACCESS_NETWORK_STATE, and ACCESS_WIFI_STATE, allow spyware to monitor network activity and transmit data. Storage access permissions, such as

READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE, and MANAGE_EXTERNAL_STORAGE (for Android 11+), enable access to files stored on the device. Additionally, device and security monitoring permissions, like READ_SMS, READ_CALL_LOG, and BIND_ACCESSIBILITY_SERVICE, provide access to sensitive user data and interactions. Advanced detection permissions, such as SYSTEM_ALERT_WINDOW and RECEIVE_BOOT_COMPLETED, allow spyware to display overlays and run persistently at startup. Finally, FOREGROUND_SERVICE enables spyware to run continuously in the background, ensuring uninterrupted data collection and transmission.

2.2.3.5. Backdoor

A backdoor is a type of malware that enables attackers to gain unauthorized remote access to a device, often with full control over its functions. On Android devices, backdoors bypass security measures to control the device without the user's knowledge. These malicious programs are particularly dangerous because they allow attackers to perform various harmful activities, such as stealing sensitive data, installing additional malware, or even taking control of hardware components like the camera or microphone. Backdoors can be implemented through various techniques, exploiting system vulnerabilities or permissions.

Android backdoors come in several forms, each posing unique threats to users. Rootkits are malware that hide their presence by modifying system files or processes. They often grant attackers "root" access, allowing them to bypass security protocols and perform actions like altering system files, installing malware, or controlling device hardware. Trojan Horses disguise themselves as legitimate apps or files but secretly provide attackers with remote access or surveillance capabilities. Delivered through phishing or malicious downloads, they often go unnoticed while performing harmful activities.

Adware and Spyware may also function as backdoors by collecting user data and transmitting it to malicious servers. Though not traditional backdoors, their data-stealing behavior can be exploited further. Custom Backdoors via APK Manipulation involve embedding malicious code into APK files, enabling attackers to remotely control the device or silently install additional malware. Lastly, Backdoors Through System Updates or Exploits take advantage of vulnerabilities in the Android operating system, allowing attackers to install backdoors via malicious updates or zero-day exploits.

Backdoor malware typically starts by exploiting Android’s permission model. Malicious apps request more permissions than necessary, such as access to the camera, microphone, SMS, or admin privileges, to carry out their activities unnoticed. Some backdoors abuse root privileges, gaining superuser access to the system. This allows them to install persistent backdoors and evade detection. Others modify system binaries, replacing core files with infected versions that enable attackers to maintain access even after a reboot.

Remote Access Tools (RATs) are another common method for implementing backdoors. These tools allow attackers to remotely control the device, execute commands, and collect sensitive data. Often installed through phishing or fake software updates, RATs operate in the background to avoid detection. Phishing and social engineering are frequently used to trick users into installing backdoor apps by disguising them as legitimate software.

Backdoor malware can severely compromise an Android system by granting attackers extensive control over the device. With root or admin access, attackers can modify or delete system files, install additional malware, and execute unauthorized commands. Backdoors can also be used to steal sensitive information, such as authentication codes, personal files, and location data, which may be transmitted to remote servers controlled by attackers.

Additionally, backdoors can enable attackers to hijack system resources, resulting in high CPU and battery usage. They can covertly activate hardware components like the camera and microphone to record users without their knowledge. Some backdoors turn infected devices into bots as part of a botnet, which can then be used for large-scale cyberattacks like Distributed Denial of Service (DDoS). These activities degrade device performance, compromise user privacy, and pose significant security risks.

Backdoors exploit Android permissions to perform malicious activities. Device Administrator Permissions allow the malware to lock the device, wipe data, or prevent uninstallation. Root Access provides superuser privileges, enabling the attacker to bypass security restrictions and access the entire operating system. SMS Permissions let backdoors read, send, and intercept messages, often stealing authentication codes or spreading malware.

Internet and Network Access Permissions allow communication with remote servers to send stolen data or receive commands. These permissions are also used to download additional malware or conduct DDoS attacks. Camera and Microphone Access enable backdoors to capture photos, videos, or conversations without user consent. Location Access can be used to track user movements in real time. Other critical permissions include Storage Access for stealing or encrypting files and Phone Management Permissions for hijacking calls or stealing call logs. These permissions collectively give attackers complete control over the device.

2.2.3.6. Worm

A worm is a type of malicious software that spreads across networks by automatically copying itself onto other computers, exploiting security flaws in the target systems. Unlike traditional viruses, which require human interaction (e.g., inserting infected disks), worms propagate independently through networks such as the Internet, making them a significant cybersecurity threat.

Worms possess distinct traits that enable their rapid spread and impact. They are self-replicating, creating multiple copies of themselves to infect other systems. Being standalone, worms do not require a host file or application to operate. Additionally, they spread automatically, requiring no user interaction to infiltrate devices or networks.

Worms operate by exploiting vulnerabilities in systems and networks to propagate. First, they identify and target devices with weak security. Once a worm infects a system, it begins replicating itself and spreads through connected networks or devices. This process continues autonomously, potentially overwhelming systems with excessive replication or bandwidth usage, enabling further malicious activities such as data theft or malware installation.

The impact of worms can range from minor disruptions to severe system damage. Worms can overload systems by consuming significant bandwidth and CPU resources, leading to slower performance or complete shutdowns. They may steal or corrupt data, compromising sensitive information. Furthermore, worms often serve as delivery systems for other malware, such as trojans or ransomware, amplifying the potential harm

Worms exploit various permissions on Android devices to facilitate their malicious activities. Network permissions such as ACCESS_WIFI_STATE and ACCESS_NETWORK_STATE allow worms to gather information about Wi-Fi connections and monitor network statuses to identify potential vulnerabilities. The INTERNET permission enables worms to communicate with remote servers, facilitating propagation and downloading additional malware. Advanced permissions like BIND_VPN_SERVICE and CAPTURE_PACKET support packet capturing and secure communication, often used for analyzing network traffic or masking malicious activities. Worms also leverage ACCESS_FINE_LOCATION to detect nearby Wi-Fi networks for exploitation. Persistent permissions such as FOREGROUND_SERVICE ensure that worms remain active in the background, while RECEIVE_BOOT_COMPLETED allows them to restart automatically when the device boots, ensuring continuity of their operations.

2.2.4. Sideloading

Sideloading refers to the process of installing applications or software onto a device from a source other than the official app store or authorized distribution platform. In computational and cybersecurity contexts, sideloading is often associated with increased risk, as it bypasses standard security checks and verification procedures. While sideloading can be useful for developers or advanced users testing custom applications, it also creates a potential attack vector for malicious actors to distribute malware, spyware, or unauthorized software. In mobile environments like Android, users may enable sideloading to install APK files manually, which makes endpoint protection and real-time threat detection critical. Understanding and managing the risks associated with sideloading is essential in the design of secure systems, especially in security-

focused applications like Patronus, where monitoring sideloaded apps plays a key role in maintaining device integrity.

2.3. Related Work

This section reviews existing research on Android malware detection and Wi-Fi security, highlighting various approaches that combine machine learning, feature extraction, and system monitoring. A number of studies have explored different techniques to enhance the accuracy, efficiency, and adaptability of malware detection models. From hybrid models combining deep learning with traditional methods to innovative solutions targeting network and Wi-Fi security, the works discussed here underscore the potential for integrating advanced AI techniques to strengthen both Android application security and network protection. Our work builds on these studies by incorporating AI models to address emerging threats, particularly in the context of sophisticated malware and Wi-Fi attack detection.

In their study, Wang et al. (2019) proposed a hybrid model combining deep autoencoder (DAE) and convolutional neural network (CNN) for Android malware detection. The model improves detection accuracy by using CNN's ability to extract features, while DAE extracts features and reduces training time. Experimental results show that the CNN-S model improves accuracy by 5% compared to traditional methods like SVM, while the DAE-CNN reduces training time by 83%. The hybrid model effectively enhances large-scale malware detection, offering significant improvements in both detection accuracy and efficiency.

In their study, Almahmoud et al. (2021) proposed a novel Android malware detection model based on a combination of static features such as permissions, API calls, and system event monitoring. The model utilizes a Recurrent Neural Network (RNN) architecture, achieving an impressive accuracy of 98.58%. The results demonstrate the superiority of this approach over traditional machine learning algorithms for detecting Android malware, highlighting its potential in improving the security of Android applications.

In their study, the developers of the FakeAP Detector app introduced a lightweight solution for detecting malicious Wi-Fi networks by analyzing static features such as signal strength (RSSI), network name (SSID), and security protocols. The app achieved high detection accuracy, with 99% in open networks and 99.7% in closed networks, but faced limitations due to its reliance on static patterns, leading to false positives. In our work, we propose enhancing this approach by integrating AI models such as Support Vector Machines (SVM), K-Nearest Neighbors (KNN), and Neural Networks. These models improve detection accuracy and adaptability by identifying sophisticated attacks, including Evil Twin, Rogue AP, and KRACK, providing more comprehensive protection against Wi-Fi threats.

Al-Ofeishat (2023) proposed a network-driven machine learning approach to enhance Android malware detection. The solution involves preprocessing network traffic data and applying feature selection techniques, such as packet size ratio and flow duration, to extract relevant network-level attributes. The methodology optimized various classifiers, including Random Forest, LGBM, and XGBoost, using techniques like RobustScaler and StandardScaler. Results showed that the Random Forest classifier achieved superior performance in terms of F2 score, precision, and recall, highlighting the effectiveness of integrating machine learning with network-based features for Android malware detection. This approach offers significant potential for real-world deployment in network security systems, with low false positive and false negative rates.

2.4. Similar Software and Comparative Analysis

This section examines existing applications that offer features similar to those in our project. By analyzing their strengths, limitations, and unique functionalities, we aim to identify gaps and opportunities that our app addresses. This comparison highlights how our solution stands out in terms of innovation, usability, and security.

2.4.1. Bitdefender Mobile Security

2.4.1.1. Overview

Bitdefender Mobile Security is an Android security suite developed by Bitdefender, a well-established leader in the cybersecurity industry. Designed to deliver comprehensive protection without compromising performance, the app offers a layered defense system that includes real-time malware scanning, behavior-based threat detection, web protection, identity monitoring, and privacy tools like app locking and anti-theft controls.

2.4.1.2. Key Features

1. Malware Scanning: Bitdefender Mobile Security includes a manual malware scanning feature that allows users to perform a full device scan to detect and remove any malware present.

2. App Anomaly Detection: An optional feature that, when enabled, continuously monitors installed apps for suspicious or malicious behavior using real-time behavior analysis.

3. Download Scan: This optional feature scans sideloaded apps *before* they are installed, providing a preemptive layer of protection against malicious APKs.

4. Web Protection: When turned on, this feature actively safeguards users from accessing dangerous or malicious websites while browsing.

5. Scam Alert: Monitors incoming texts, app notifications, and chat messages for malicious links, helping detect phishing and scam attempts and prevent the spread of attacks via shared URLs.

6. Privacy Features: Includes tools such as App Lock and Anti-Theft, designed to protect sensitive apps and data, and secure the device in the event of loss or theft.

2.4.1.3. Key Strengths

1. Performance Optimization Through Customizable Settings: Bitdefender Mobile Security allows users to fine-tune which features run in the background, giving them control over resource usage. This balance between protection and performance makes the app especially suitable for lower-end or older devices without compromising core security.

2. High Privacy: While the app scans sensitive content such as chat messages (for Scam Alert) and user-installed apps (for malware), Bitdefender is transparent about its practices. The UI clearly communicates that scans are local and limited to harmful content detection, with data collection restricted to what's outlined in their privacy policy.

3. User Friendly UI: Despite packing a wide range of security features, Bitdefender Mobile Security maintains a clean, intuitive interface. Navigation is straightforward, and the app uses clear, non-technical language, making it accessible even for users with minimal cybersecurity knowledge.

2.4.1.4. Key Weaknesses

1. Security Features Auto-disabling: Numerous users have reported that core security features—particularly *Web Protection*—are frequently disabled without warning and must be manually reactivated. This recurring issue suggests that the app may prioritize performance over continuous protection. Given how long the problem has persisted, it appears more like an intentional design choice than a software bug, raising concerns about reliable threat coverage.

2. False Positive Spam: When Bitdefender flags a benign app as malicious, it tends to bombard users with repeated warnings, with no apparent way to suppress or whitelist the detection. This creates a frustrating user experience that, according to several reviews, has led users to uninstall the app entirely—ironically removing any protection altogether.

3. Dangerous False Negatives: A particularly critical concern raised by users is the app's failure to detect some high-risk malware, including ransomware. These false negatives pose a significant security risk and may be linked to the auto-disabling of detection features.

4. App Lock Reliability Issues: The *App Lock* feature appears to be inconsistent. Users report that it works initially but eventually stops prompting for a passcode, effectively leaving protected apps unlocked after a few days. This undermines its intended purpose and creates a false sense of security.

2.4.1.5. Bitdefender Mobile Security User Interface

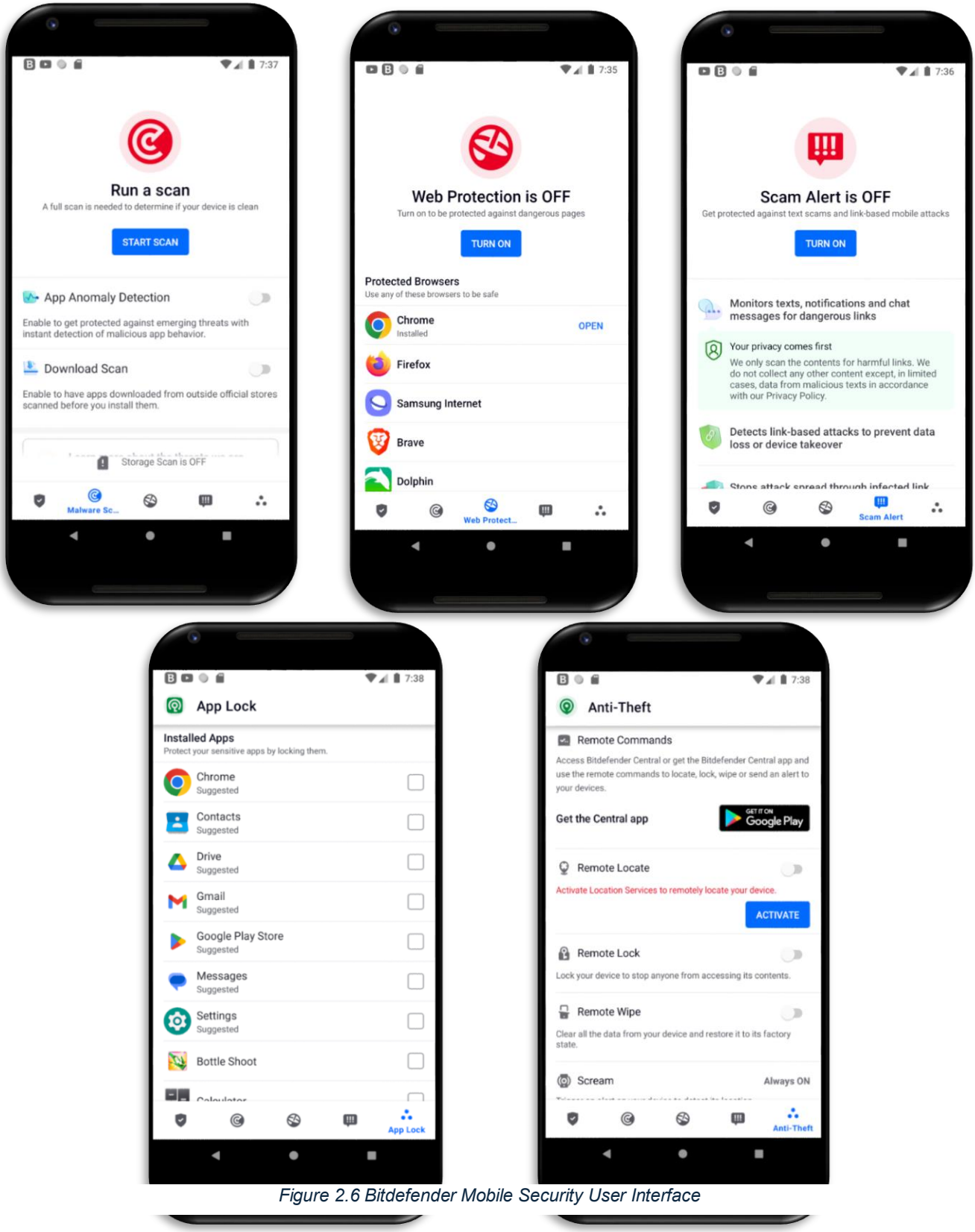

Figure 2.6 Bitdefender Mobile Security User Interface

2.4.1.6. Comparative Analysis of Bitdefender and Patronus

Table 2.1 Comparative Analysis of Bitdefender and Patronus

Features	Bitdefender	Patronus
Malware Scanning	✓	✓
App Anomaly Detection	✓	✗
Download Scan	✓	✓
Web Protection	✓	✓
Scam Alert	✓	✗
Privacy Features	✓	✗
Cost	Free + Premium	Free

2.4.2. Avast Antivirus & Security

2.4.2.1. Overview

Avast Antivirus & Security is an app designed to protect Android and iOS devices from various digital threats like malware phishing, and hackers. It offers both free and premium versions with different features aimed improving the security, privacy, and performance of mobile devices.

2.4.2.2. Key Features

1. **Malware Protection:** Scans apps, files, and websites to detect and block malware, viruses and other malicious threats.
2. **Web Protection:** Blocks unsafe websites and phishing attempts, protecting users while browsing the web.
3. **Wi-Fi Security:** Analyzes Wi-Fi networks to detect vulnerabilities, ensuring safer internet connections, especially on public Wi-Fi.
4. **Anti-Theft Tools:** Allows remote tracking, locking, or wiping of the device if it's lost or stolen.
5. **App Lock:** Secures individual apps with a PIN, pattern, or fingerprint lock to prevent unauthorized access.
6. **Junk Cleaner:** Helps clean up unnecessary files and cache, improving the device's performance.
7. **VPN:** Offers secure, private browsing by encrypting your internet connection.

2.4.2.3. Key Strengths

1. **Frequent Updates:** Avast is known for regularly updating its virus definitions, meaning it stays up-to-date with the latest malware and security threats.

2. **Customizable Scans:** Users can customize how Avast scans their device, choosing between quick scans, deep scans, or specific folder or app scans. This flexibility allows for more targeted protection.
3. **User-Friendly Interface:** Avast offers an intuitive interface that is relatively easy to navigate, even for users who are not tech-savvy.
4. **Community and Support:** Avast has a robust user community and offers support resources, helping users resolve issues quickly.
5. **Privacy Advisor:** The Privacy Advisor analyzes installed apps and tells users which apps have access to sensitive information. This can help users understand which apps are potentially invasive and adjust their privacy settings accordingly.
6. **Convenient Anti-Theft Features:** The ability to remotely locate, lock, or wipe a device provides peace of mind for users concerned about theft.
7. **Wide Range of Features in One App:** Avast offers an all-in-one security solution that combines malware protection, anti-theft, privacy controls, a junk cleaner, VPN, and more, making it a versatile tool for mobile security without needing multiple separate apps.
8. **User Trust and Reputation:** Avast has established a strong reputation in the cybersecurity field, with millions of satisfied users worldwide.

2.4.2.4. Key Weaknesses

1. **Performance Impact:** Running continuous scans and real-time protection can slow down the device, particularly on older or lower-end smartphones. Background processes may consume system resources, leading to slower performance or faster battery drain.
2. **VPN Limitations:** While Avast includes a VPN feature in its premium package, it's often limited in speed and server locations compared to standalone VPN services. Additionally, users might experience slower internet connections when using the VPN.
3. **False Positives:** Like many antivirus apps, Avast can sometimes flag legitimate apps or files as threats, causing unnecessary alarm or interruptions to normal usage.
4. **Root Dependency for Firewall:** The firewall feature is only available on rooted Android devices, which limits its accessibility to users who don’t want to or cannot root their devices. Rooting a device can also void warranties and expose it to other security vulnerabilities.
5. **Subscription Cost:** While the free version provides basic protection, the premium subscription can be relatively expensive for some users, especially considering the competition in the mobile security market, where many providers offer similar services at lower prices or for free.
6. **Battery Usage:** Some users report higher battery consumption due to the app’s background activity, especially with features like real-time protection, Wi-Fi scanning, and anti-theft features constantly running.

7. **Data Usage:** Avast Mobile Security might consume a significant amount of data for activities like real-time scanning, updates, and cloud-based threat detection. This can be a concern for users on limited data plans.

2.4.2.5. Avast Antivirus & Security User Interface

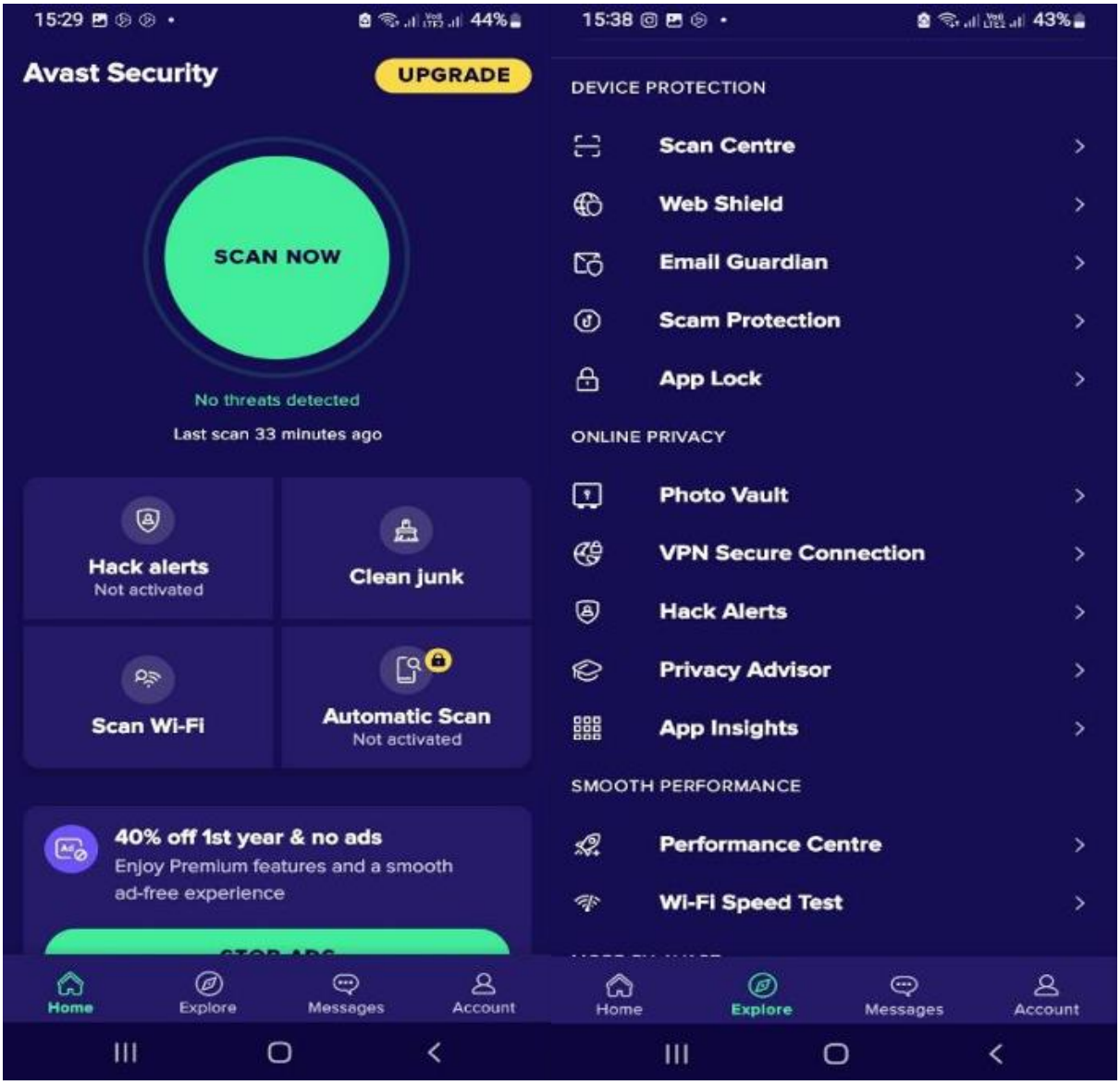

Figure 2.7 Avast Antivirus & Security User Interface

2.4.2.6. Comparative Analysis of Avast and Patronus

Table 2.2 Comparative Analysis of Avast and Patronus

Features	Avast	Patronus
Malware Protection	✓	✓
Web Protection	✓	✓
Wi-Fi Security	✓	✓
Anti-Theft Tools	✓	✗
App Lock	✓	✗
Junk Cleaner	✓	✗
VPN	✓	✗
Cost	Free + Premium	Free

2.4.3. McAfee Security

2.4.3.1. Overview

It is an application designed to protect Android and iOS devices from a variety of threats, including malware, viruses, phishing attacks, and privacy breaches. Developed by McAfee Corp. This app aims to provide users with secure mobile experience.

2.4.3.2. Key Features

1. **Anti-Malware Protection:** McAfee offers strong anti-malware protection by scanning for and blocking viruses, malware, and suspicious apps in real time. This ensures that users are shielded from both known and emerging threats as they arise.
2. **Wi-Fi Security:** The software evaluates the safety of public Wi-Fi networks and alerts users if a connection is insecure or potentially compromised. This feature helps users avoid risky networks that could lead to data theft or intrusion.
3. **App Privacy Check:** McAfee includes a privacy auditing feature that reviews app permissions on the device. It identifies apps that access sensitive data and highlights potential privacy risks, giving users better control over their personal information.
4. **Anti-Theft:** With remote tracking capabilities, McAfee enables users to locate their lost or stolen devices. Additionally, it allows users to remotely lock or wipe their device to protect personal data from unauthorized access.
5. **Web Protection:** McAfee enhances web safety by blocking malicious websites and protecting users from phishing attacks. This feature actively monitors browsing activity and prevents access to dangerous content.

6. **VPN:** A built-in Virtual Private Network (VPN) encrypts internet traffic to provide secure and private browsing, especially useful when connected to public Wi-Fi networks. This prevents third parties from intercepting user data.
7. **Identity Protection:** McAfee monitors for data breaches and alerts users if their personal information, such as email addresses or passwords, has been exposed online. This helps users take timely action to secure their identities.

2.4.3.3. Key Strengths

1. **Comprehensive Protection:** McAfee delivers well-rounded security through real-time malware detection and web protection. It continuously scans for viruses and suspicious activity, while also blocking harmful websites and alerting users to phishing attempts.
2. **Anti-Theft Features:** To help recover lost devices, McAfee includes powerful anti-theft tools such as remote locking, location tracking, and data wiping. It also offers a siren feature that allows users to sound an alarm on their device to aid in locating it.
3. **Privacy Protection:** The app provides robust privacy tools, including detailed reports on app behavior and permission usage. In addition, its secure browsing features—such as VPN access—help protect sensitive data when users connect to unsecured networks.
4. **Performance Optimization:** McAfee helps maintain device efficiency with tools that clean up junk files and manage storage. It also includes battery optimization features that extend device life by controlling background processes and unnecessary power usage.
5. **Cross-Platform Support:** Designed for versatility, McAfee is compatible with multiple operating systems, including Android, iOS, Windows, and macOS. This allows users to enjoy a consistent level of security across all their devices.
6. **Frequent Updates:** To ensure protection against the latest threats, McAfee regularly updates its virus definitions and security features. These frequent updates help maintain its effectiveness against newly emerging cyber risks.

2.4.3.4. Key Weaknesses

1. **High Resource Usage:** McAfee can be resource-intensive, leading to slower device performance, especially on older or less powerful devices. Users often report lag and slowdowns when the app runs in the background.
2. **Battery Drain:** The app's continuous scanning and real-time protection features can significantly drain battery life, which is a common complaint among users.
3. **Costly Premium Version:** While it offers a free version, many essential features are locked behind a premium subscription. The subscription cost may be considered high compared to competitors that offer similar or better services.
4. **Frequent Notifications:** Users often receive numerous alerts and notifications, which can be overwhelming and lead to notification fatigue. Some find it annoying to manage constant prompts from the app.

5. **Limited features in the basic version:** The free version offers basic scanning but lacks comprehensive features like anti-theft tools and advanced privacy protection, which can be a letdown for users looking for robust free options.
6. **Customer Support Issues:** Some users have reported difficulties in reaching customer support or receiving timely assistance, which can be frustrating if they encounter issues with the app.
7. **Mixed Reviews on Malware Protection:** While McAfee claims to protect against malware, reviews are mixed. Some users have experienced infections even with McAfee installed. This raises concerns about its effectiveness.
8. **Compatibility Issues:** Compatibility is another concern. Some users report issues with certain devices. McAfee may not work well with older systems.

2.4.3.5. McAfee Security User Interface

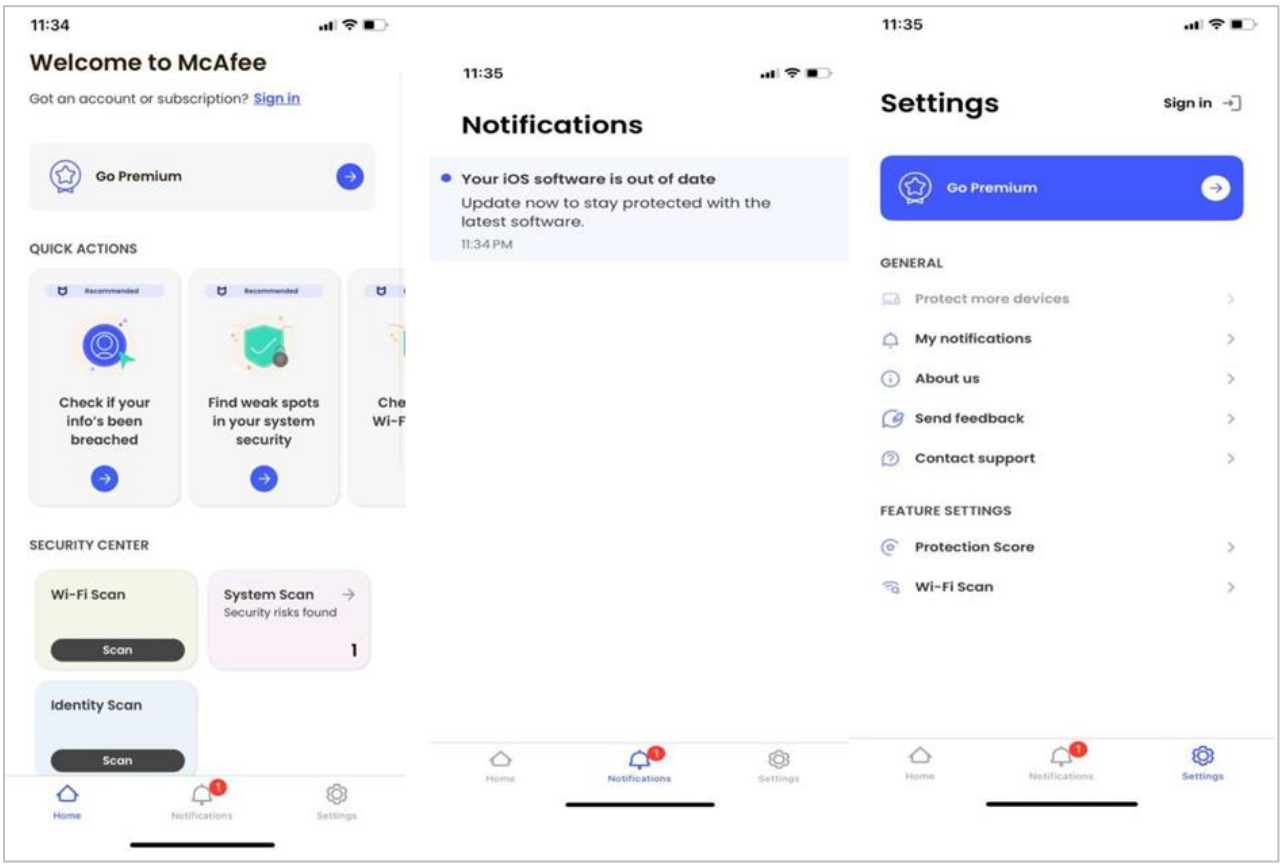

Figure 2.8 McAfee Security User Interface

2.4.3.6. Comparative Analysis of McAfee and Patronus

Table 2.3 Comparative Analysis of McAfee and Patronus

Features	McAfee	Patronus
Anti-Malware Protection	✓	✓
Wi-Fi Security	✓	✓
App Privacy Check	✓	✘
Anti-Theft	✓	✘
Web Protection	✓	✓
VPN	✓	✘
Identity Protection	✓	✘
Cost	Free + Premium	Free

2.4.4. Fing – Network Tools

2.4.4.1. Overview

Fing is a popular network scanner app designed to help users monitor and manage their home networks. It enables users to identify connected devices, detect unknown ones, and diagnose network issues while providing an overview of network status and potential security vulnerabilities. However, Fing lacks advanced AI-based threat analysis found in other security applications and primarily focuses on network diagnostics, offering detailed device recognition and monitoring tools rather than actively preventing security threats, especially on public networks.

2.4.4.2. Key Features

1. **Network Scanning & Device Detection:** Fing identifies devices connected to a home network, providing details such as IP addresses, MAC addresses, and device types.
2. **Internet Speed Test:** The app tests internet speed, giving real-time download and upload speeds, helping users monitor network performance.
3. **Network Security Assessment:** Fing checks for open ports, weak security settings, and other potential vulnerabilities within the home network.
4. **Network Troubleshooting Tools:** Fing offers tools to diagnose network issues, such as IP conflicts and slow performance.

2.4.4.3. Key Strengths

1. **Device Identification:** Fing allows users to easily identify all devices connected to their network, providing detailed information about each device, such as IP address, MAC address, and device type.

2. **Network Monitoring:** It offers real-time monitoring of your network, helping you detect any unauthorized or unknown devices that may have accessed your network.
3. **Network Health Checks:** Fing performs network diagnostics and provides insights into potential performance issues, such as slow connections or weak signals, helping users troubleshoot problems efficiently.
4. **User-Friendly Interface:** The app is easy to use, making it accessible for both tech-savvy individuals and those with limited networking knowledge.
5. **Security Insights:** Fing alerts users to possible security vulnerabilities and offers an overview of the network's overall security posture.
6. **Free and Paid Features:** The app offers both free and premium features, with the premium version providing additional advanced tools like network speed testing and more in-depth monitoring capabilities.
7. **Cross-Platform Availability:** Fing is available on multiple platforms, including iOS, Android, and web, providing flexibility for users to access and manage their network from anywhere.

2.4.4.4. Key Weaknesses

1. **Limited Public Wi-Fi Protection:** While Fing offers great insights for home networks, it doesn't offer protection against public Wi-Fi threats like rogue access points, man-in-the-middle attacks, or SSL stripping, which are crucial in unsecured environments.
2. **No AI-Based Detection:** Fing's static threat detection methods lack the adaptability and depth of AI-powered real-time monitoring. It can only report basic network conditions without offering automated preventive measures or in-depth risk analysis.
3. **No Automatic Response:** Unlike our app, which automatically disconnects from unsafe networks, Fing merely informs users about potential vulnerabilities without providing direct actions.
4. **No Malware Detection:** Fing focuses solely on network scanning and device identification. It lacks integrated malware protection, leaving users exposed to threats delivered through public networks.
5. **Premium Features Locked Behind Paywall:** Some advanced features, such as detailed network analysis and historical data, are only available in the premium version, limiting the functionality of the free app.
6. **Device Identification Limitations:** Struggles to accurately identify newer or less common devices.
7. **No Vulnerability Assessment:** Doesn't assess weak passwords, outdated firmware, or open ports that could be exploited.
8. **Lack of Privacy Features:** Does not protect against network-level tracking or data leaks.
9. **No Network Behavior Analysis:** Doesn't analyze network activity for suspicious patterns or attacks.

10. **No Proactive Security Recommendations:** Lacks AI-powered suggestions to improve network security based on real-time analysis.

2.4.4.5. Fing User Interface

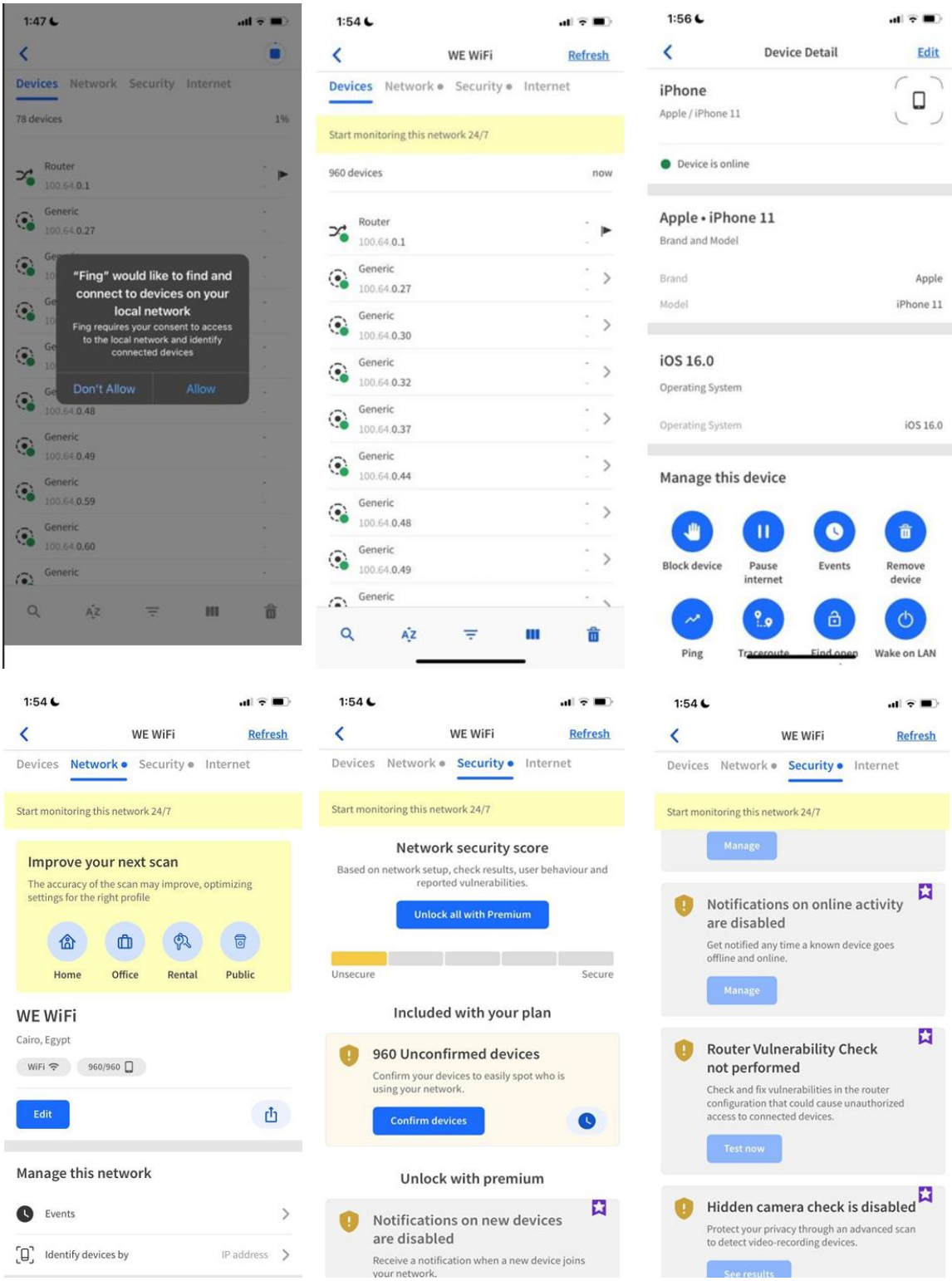

Figure 2.9 Fing User Interface

2.4.4.6. Comparative Analysis of Fing and Patronus

Table 2.4 Comparative Analysis of Fing and Patronus

Features	Fing	Patronus
Network Scanning & Device Detection	✓	✓
Internet Speed Test	✓	✓
Network Security Assessment	✓	✓
Network Troubleshooting Tools	✓	✓
Cost	Free + Premium	Free

2.4.5. Trend Micro Mobile Security

2.4.5.1. Overview

Trend Micro Mobile Security is an all-in-one mobile protection app that safeguards Android and iOS devices from malware, phishing, unsafe websites, and network threats. It features real-time malware scanning, web protection, Wi-Fi security, a built-in VPN for safe browsing, and parental controls through content filtering. Designed for privacy and performance, it also includes anti-theft tools and system optimization. While it offers strong overall protection, some users report issues such as slowed internet speeds, limited content filtering, and lack of detailed activity logs.

2.4.5.2. Key Features

1. Malware Protection: Scans apps and files for viruses, trojans, ransomware, and spyware. Uses cloud-based intelligence for up-to-date protection.

2. Web Protection: Blocks malicious websites, phishing links, and unsafe URLs in browsers, social media, and messaging apps.

3. Wi-Fi Checker: Scans Wi-Fi networks for security issues like MITM (Man-in-the-Middle) attacks or weak encryption.

4. App Privacy Scanner: Analyzes installed apps to identify those that access sensitive information (location, contacts, camera, etc.).

5. Parental Controls: Includes tools to restrict app access, monitor child usage, and block inappropriate content.

6. System Tuner: Helps optimize memory usage and improve battery life by closing background apps.

7. Anti-Theft: Includes remote lock, wipe, locate, and alarm features in case the device is lost or stolen.

8. Pay Guard: A secure browser that protects online banking and financial transactions.

9. Identity Protection: Alerts you if your personal information (email, password) is found in a data breach.

2.4.5.3. Key Strengths

1. All-in-one Protection: Covers malware, phishing, web threats, privacy, and performance from one app.

2. User-Friendly Interface: Simple and easy for non-technical users to understand and manage.

3. Frequent Updates: Trend Micro's cloud-based threat intelligence ensures the app is always updated.

4. Low Battery Consumption: Lightweight design with minimal impact on device performance and battery life.

5. Cross-platform Sync: Can sync and monitor multiple devices under the same Trend Micro account.

6. Real-time Scanning: Immediate alerts and scanning when new apps or files are downloaded.

2.4.5.4. Key Weaknesses

1. Some Features Require Subscription: Full protection, such as Pay Guard and advanced anti-theft, requires a premium license.

2. Limited Behavioral Analysis: Relies more on signature-based detection than advanced AI-based behavioral analysis.

3. Unreliable Parental Controls: Parental controls were found to be ineffective in blocking certain adult content, leading to concerns about its reliability.

4. VPN Interference with Browser: Safari and other browsers failed to connect to websites or follow links when VPN was enabled by the app.

5. Ineffective Content Filtering: The adult content filter failed to block inappropriate content, even with the child setting enabled.

6. Wi-Fi Speed Slowdown: Significant reduction in Wi-Fi connection speeds after app installation (from 400+ Mbps to 10-20 Mbps).

7. Absence of Activity Logs: The app does not provide a detailed log of blocked activities or security events, making it difficult for users to monitor and track the app's actions and effectiveness.

2.4.5.5. Trend Micro Mobile Security User Interface

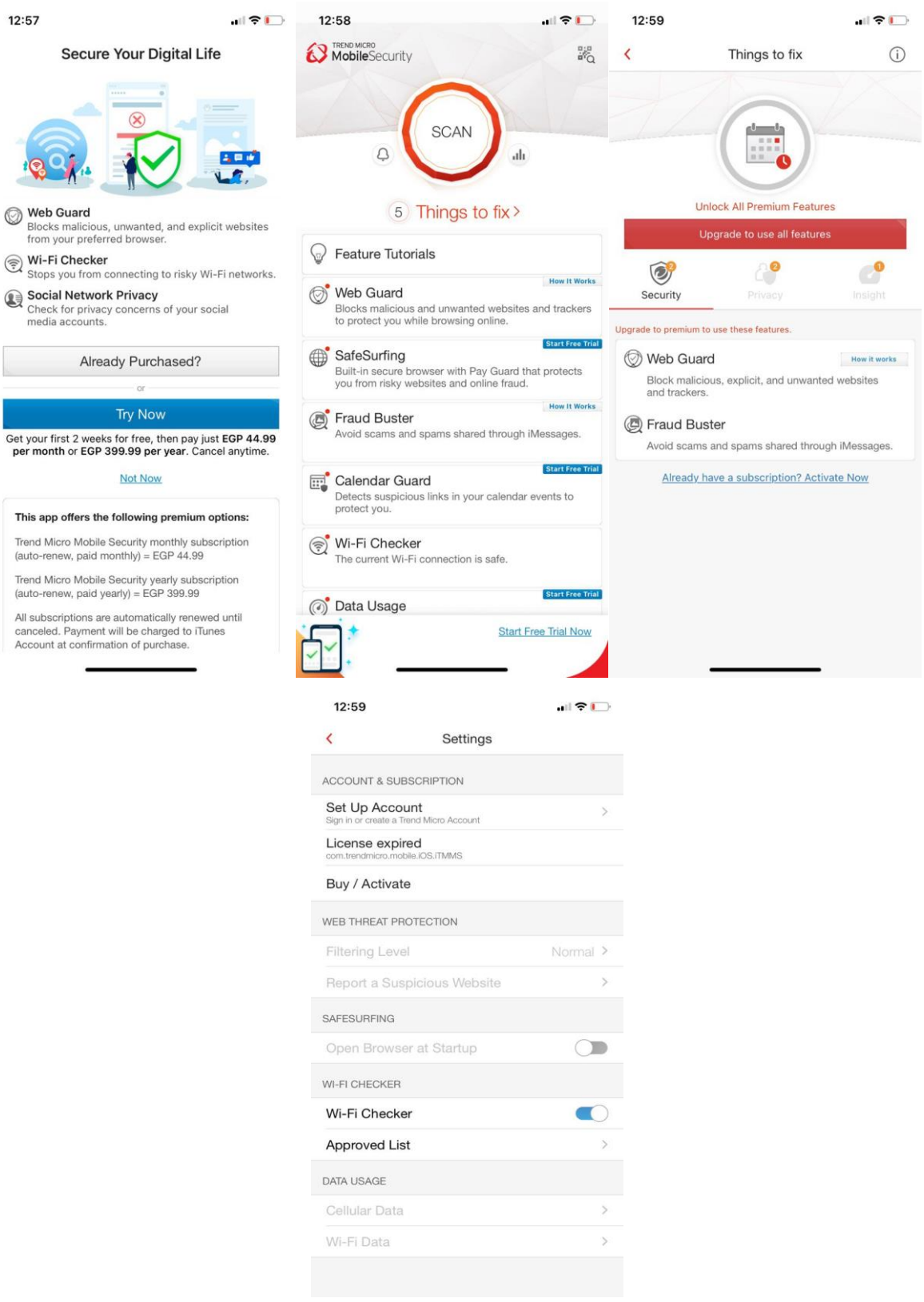

Figure 2.10 Trend Micro Mobile Security User Interface

2.4.5.6. Comparative Analysis of Trend Micro and Patronus

Table 2.5 Comparative Analysis of Trend Micro and Patronus

Features	Trend Micro	Patronus
Malware Protection	✓	✓
Web Protection	✓	✓
Wi-Fi Checker	✓	✓
App Privacy Scanner	✓	✘
Parental Controls	✓	✘
System Tuner	✓	✘
Anti-Theft	✓	✘
Pay Guard	✓	✘
Identity Protection	✓	✘
Cost	Premium	Free

2.4.6. Kaspersky Mobile Device Security

2.4.6.1 Overview

Kaspersky Mobile Device Security provides basic protection for Android and iOS devices, including antivirus scanning, app locking, and anti-theft features. It offers additional tools like VPN and data leak monitoring, with options for individual and business use. The solution integrates with some enterprise management systems but can sometimes affect device performance during scans. Overall, it covers essential mobile security needs but may have limitations depending on user expectations.

2.4.6.2. Key Features

1. **Real-Time Antivirus:** Continuously scans the device to detect and block malware and malicious apps.
2. **Anti-Phishing Protection:** Blocks access to fraudulent websites designed to steal personal information.
3. **App Lock:** Allows users to secure specific apps with a PIN or biometric lock to prevent unauthorized access.
4. **Anti-Theft Tools:** Enables remote tracking, locking, and wiping of the device if it is lost or stolen.
5. **VPN (Virtual Private Network):** Encrypts internet traffic to protect user privacy, especially on public Wi-Fi networks.

6. **Data Leak Checker:** Monitors if your personal data has been compromised or exposed in data breaches.

2.4.6.3. Key Strengths

1. **Comprehensive Protection Suite:** Combines antivirus, anti-phishing, anti-theft, and app lock features in one app.
2. **Data Leak Monitoring:** Alerts users about data breaches involving their personal information.
3. **Smart Device Detection:** Help identify unauthorized devices connected to your home Wi-Fi network.
4. **App-Level Security Control:** The App Lock feature enables users to secure individual applications using PINs or biometrics, adding a layer of control over sensitive data.

2.4.6.4. Key Weaknesses

1. **Performance Overhead:** Device performance may degrade during real-time scanning or background operations.
2. **Limited Advanced Controls for Power Users:** May not offer deep configuration options required by security professionals.
3. **Buggy VPN and Phishing Protection:** VPN may show connection issues or desync between system and app status; phishing protection sometimes disables itself or causes errors after rebooting.
4. **Cluttered and Confusing Interface:** The app suffers from poor user experience, with hard-to-navigate settings and inconsistent notification visibility.
5. **Battery Consumption:** Continuous background activity can drain the battery faster on certain devices.
6. **Missing Features on iOS:** Features like on-demand scanning, firewall, and full browser protection are absent or limited, especially on Apple devices.

2.4.6.5. Kaspersky Mobile Device Security User Interface

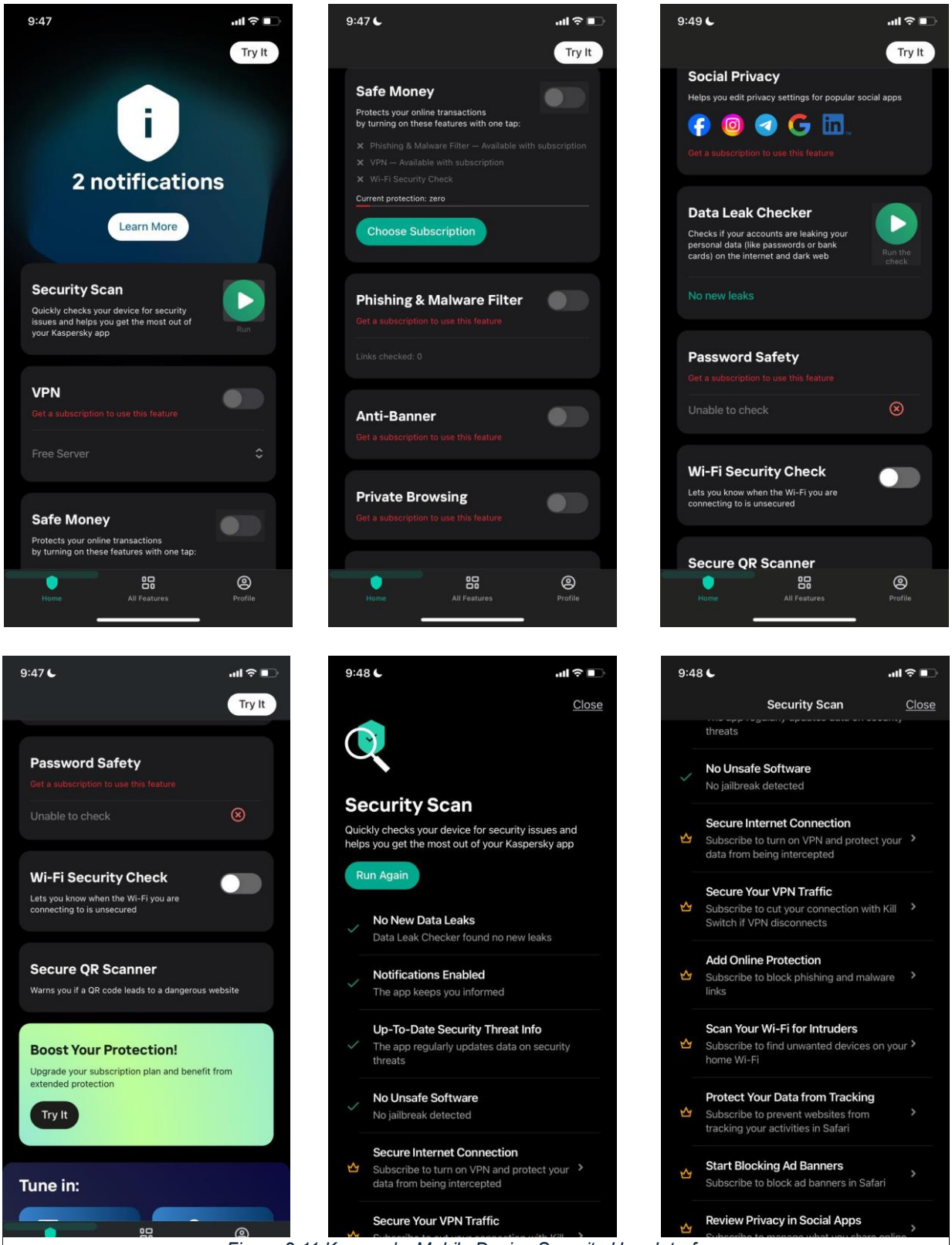

Figure 2.11 Kaspersky Mobile Device Security User Interface

2.4.6.6. Comparative Analysis of Kaspersky and Patronus

Table 2.6 Comparative Analysis of Kaspersky and Patronus

Features	Kaspersky	Patronus
Real-Time Antivirus	✓	✓
Anti-Phishing Protection	✓	✓
App Lock	✓	✘
Anti-Theft Tools	✓	✘
VPN	✓	✘
Data Leak Checker	✓	✘
Cost	Free + Premium	Free

2.4.7. Sophos Intercept X for Mobile

2.4.7.1. Overview

Sophos Intercept X for Mobile is a robust security application that safeguards mobile devices against a wide range of threats, including malware, phishing attacks, and network vulnerabilities. Leveraging the same deep learning anti-malware technology found in Sophos' desktop solutions, it ensures consistent protection across all platforms.

2.4.7.2. Key Features

1. Malware Protection: Scans installed apps for malicious behavior or reputation issues. Uses real-time threat intelligence from SophosLabs.

2. Web Filtering: Blocks access to known malicious or inappropriate websites using Sophos web filtering technology.

3. App Reputation: Assesses the trustworthiness of apps based on community reputation and known threat data.

4. Device Compliance Checks: Ensures the device adheres to corporate security policies (e.g., encryption, rooting status, OS version).

5. Network Protection: Detect suspicious or insecure Wi-Fi connections and prevent MITM attacks.

6. Phishing Protection: Identifies and blocks phishing URLs in real-time through link scanning.

7. Privacy Advisor: Flags apps with excessive permissions or privacy risks.

8. Centralized Management: Admins manage policies, monitor alerts, and view compliance status via Sophos Central.

9. Battery Optimization: Designed to run in the background with minimal resource consumption.

2.4.7.3. Key Strengths

1. **Comprehensive Threat Detection:** Blocks malware, phishing, malicious Wi-Fi, and risky apps in real-time.
2. **Cross-Platform Coverage:** Works on both Android (full security) and iOS (limited due to restrictions).
3. **Centralized Management:** Businesses control all devices from one dashboard (Sophos Central).
4. **Privacy & App Risk Analysis:** Scans apps for invasive permissions and hidden risks.
5. **Low Performance Overhead:** Lightweight protection with minimal battery drain.
6. **Network & Web Threat Protection:** Blocks unsafe Wi-Fi and malicious websites automatically.

2.4.7.4. Key Weaknesses

1. **Limited Malware Remediation (on iOS):** Due to Apple's sandboxing, **iOS protection is limited** to detecting risky configurations and URLs; it cannot scan apps or files.
2. **Reliance on Sophos Ecosystem:** Full features are best utilized within the **Sophos Central** ecosystem; integration with non-Sophos tools may be less robust.
3. **No VPN or In-App Threat Isolation:** Lacks a built-in secure VPN or containerized environment for high-security scenarios (e.g., separating work/personal data).
4. **Phishing Protection Gaps:** Phishing protection may depend on user permissions or browser support; some mobile browsers may bypass scanning.
5. **Basic Reporting for Small Deployments:** Reporting in Sophos Central may be limited for small businesses without premium licensing tiers.

2.4.7.5. Sophos Intercept X for Mobile User Interface

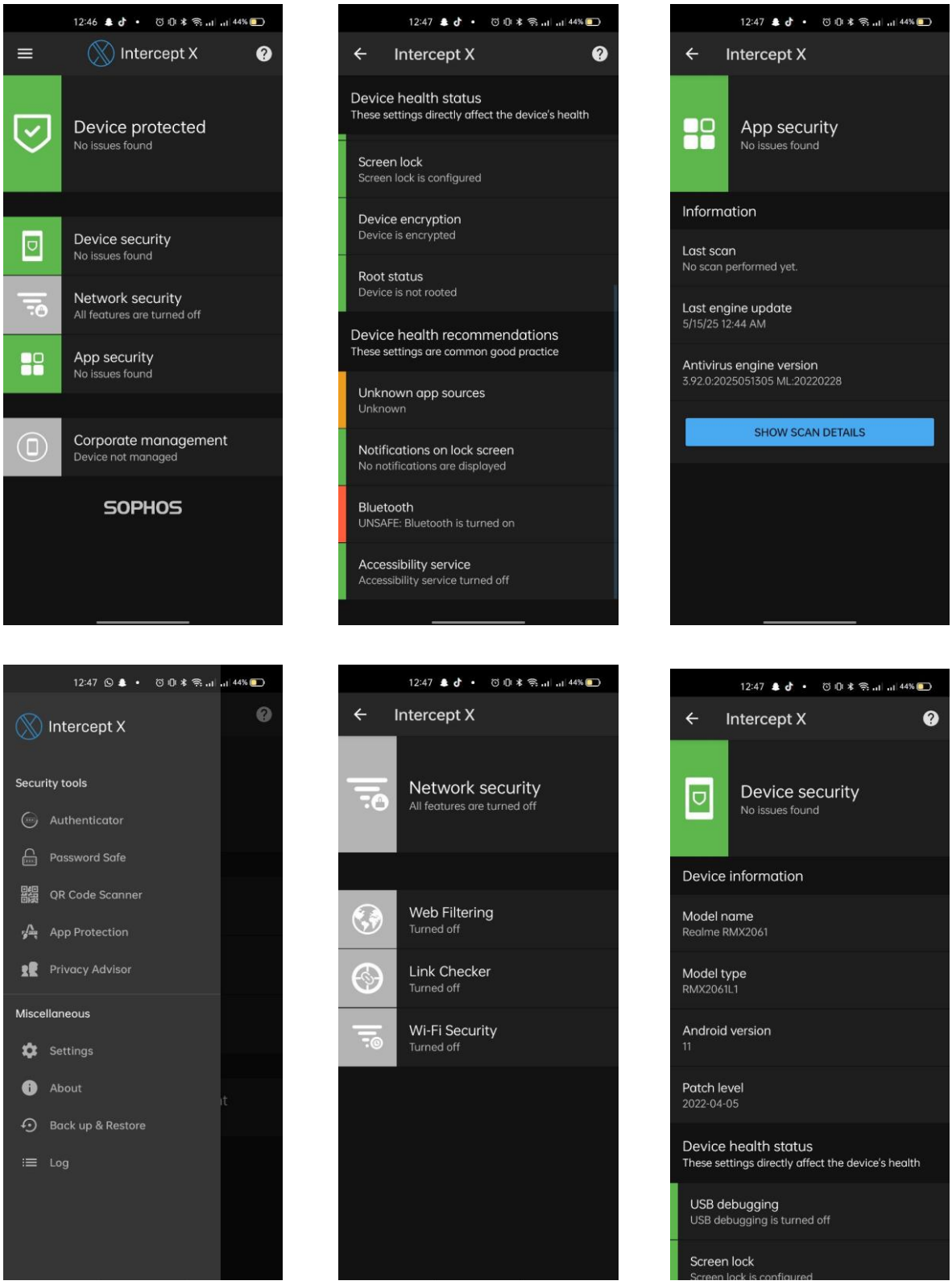

Figure 2.12 Sophos Intercept X for Mobile User Interface

2.4.7.6. Comparative Analysis of Sophos and Patronus

Table 2.7 Comparative Analysis of Sophos and Patronus

Features	Sophos	Patronus
Malware Protection	✓	✓
Web Filtering	✓	✓
App Reputation	✓	✘
Device Compliance Checks	✓	✘
Network Protection	✓	✓
Phishing Protection	✓	✓
Privacy Advisor	✓	✓
Centralized Management	✓	✘
Battery Optimization	✓	✓
Cost	Free + Premium	Free

2.4.8. Norton Mobile Security

2.4.8.1. Overview

Norton Mobile Security is a comprehensive security solution designed to protect smartphones and tablets from malware, phishing, theft, and other digital threats. Developed by NortonLifeLock (now part of Gen Digital), it offers multi-layered protection for Android and iOS devices, ensuring privacy and security in an increasingly mobile-dependent world.

2.4.8.2. Key Features

1. **Malware Protection:** Scans apps and files for viruses, spyware, and other malicious software.
2. **Wi-Fi Security:** Alerts you about unsafe Wi-Fi networks and helps prevent man-in-the-middle attacks.
3. **Web Protection:** Blocks malicious websites and phishing scams.
4. **App Advisor:** Checks apps for privacy risks before you download them.
5. **Anti-Theft Tools:** Includes remote lock, locate, and wipe features in case your device is lost or stolen.
6. **Call & Text Filtering:** Blocks spam calls and SMS messages.
7. **Dark Web Monitoring (in some versions):** Alerts you if your personal info is found on the dark web.

2.4.8.3. Key Strengths

1. **Strong Malware & Virus Protection:** Real-time scans block malware, spyware, and ransomware with frequent updates.
2. **Wi-Fi Security & VPN:** Scans Wi-Fi for attacks and includes a VPN (premium) for encrypted browsing.
3. **Anti-Theft & Device Recovery:** Remote lock/wipe, GPS tracking, and SIM change alerts.
4. **Web Protection & Phishing Defense:** Blocks dangerous websites and scam links in emails/messages.
5. **App Advisor:** Scans apps for privacy risks and excessive permissions before/after install.
6. **Call & SMS Filtering (Android):** Stops spam calls and texts automatically.
7. **Dark Web Monitoring (Premium):** Alerts if your personal data is found on the dark web.
8. **Lightweight & User-Friendly:** Minimal performance impact with a simple interface.
9. **Trusted Brand with Support:** Norton-backed security with 24/7 help for paid users.

2.4.8.4. Key Weaknesses

1. **Limited Free Version:** Most advanced features like VPN and dark web monitoring require payment.
2. **Weaker iOS Protection:** iPhone version lacks real-time scanning and call filtering due to Apple's restrictions.
3. **Performance Impact:** May slow down devices or drain battery during intensive scans.
4. **Basic VPN:** Included VPN has fewer servers and slower speeds than premium VPN services.

2.4.8.5. Norton Mobile Security User Interface

Figure 2.13 Norton Mobile Security User Interface

2.4.8.6. Comparative Analysis of Norton and Patronus

Table 2.8 Comparative Analysis of Norton and Patronus

Features	Norton	Patronus
Malware Protection	✓	✓
Wi-Fi Security	✓	✓
Web Protection	✓	✓
App Advisor	✓	✓
Anti-Theft Tools	✓	✘
Call & Text Filtering	✓	✘
Dark Web Monitoring (in some versions)	✓	✘
Cost	**Premium**	**Free**

Table 2.9 Comparative Analysis Summary

Features/Apps	Bitdefender	Norton	Fing	Kaspersky	Sophos	Avast	McAfee	Trend Micro	Patronus
AI-Powered Malware Protection	✓	✘	✘	✓	✓	✘	✓	✓	✓
Application-Level Network Threat Detection	✘ (limited to browsers only)	✘ (limited to browsers only)	✘	✘	✘	✘ (limited to browsers only)	✘ (limited to browsers only)	✘ (limited to browsers only)	✓
Wi-Fi Threat Detection and Risk Assessment	✘	✓	✓	✓	✓	✓	✓	✓	✓
Wi-Fi Speed Test	✘	✘	✓	✘	✘	✘	✘	✘	✓
Wi-Fi Troubleshooting	✘	✘	✓	✘	✘	✘	✘	✘	✓
Customizable Security Modes	✓	✘	✘	✘	✘	✘	✘	✘	✓
Chatbot	✓	✘	✘	✘	✘	✘	✘	✘	✓

2.6. Conclusion

In this chapter, we covered foundational concepts essential for understanding the rest of the project, particularly within the conceptual framework. We also reviewed related research to establish the academic basis of our work. Finally, we analyzed similar software solutions, highlighting their key features, strengths, and limitations, and conducted a comparative evaluation to show how they differ from and relate to Patronus.

Chapter 3

Dataset Preparation and AI Algorithms

3.1. Introduction

This chapter focuses on the data-centric and algorithmic aspects of Android malware detection. It begins with an overview of the datasets used for training and evaluating models, including DREBIN, TUANDROMD, and COLCOM. It then covers feature engineering techniques with an emphasis on permission-based indicators of malicious behavior. A detailed examination of neural network models and their generalization capabilities follows. Finally, this chapter introduces the integration of large language models (LLMs), presenting their architecture and potential use as APIs for improving analysis and detection in mobile security systems.

3.2. Datasets Overview

3.2.1. DREBIN Dataset

The DREBIN dataset contains approximately **15,000 Android application samples**, labeled for malware detection. It distinguishes between **malicious (S)** and **benign (B)** applications. This dataset is considered to represent **older malware**, capturing threats that were prevalent during its collection period. It is widely used in academic research for evaluating static analysis techniques and Android malware classification.

3.2.2. TUANDROMD Dataset

The TUANDROMD dataset consists of around **4,000 Android apps**, divided into **malware** and **goodware** classes. It is designed to reflect **newer malware trends**, offering a more recent snapshot of Android security threats.

3.2.3. COLCOM Dataset

The COLCOM dataset includes about **400 Android samples**, labeled with **1 for malware** and **0 for benign** applications. It is temporally situated **between DREBIN and TUANDROMD**, representing **malware that is newer than that in DREBIN but older than TUANDROMD**. Despite its smaller size, it can be valuable for evaluating model generalization across different malware generations.

3.3. Feature Engineering

3.3.1. Overview

In Android malware detection, understanding how different types of malwares interact with system permissions is essential for effective feature engineering. Each malware type typically relies on a specific set of permissions to carry out its malicious operations. While the presence of a single permission does not definitively indicate that an application is malicious, the combination of certain permissions can be a strong indicator of malicious behavior. As a result, using raw permission features alone may not be sufficient for robust malware detection. Instead, incorporating **derived or engineered permission features**—which capture meaningful patterns or combinations—can enhance model generalization and detection performance.

3.3.2. Android Malware Permission Profile

In the following section, we analyze the most commonly requested permissions by various Android malware types. This analysis is based on insights gathered from research papers, articles, and our own assessments of which permissions are typically required to execute specific malicious functions.

- Ransomware
- Spyware
- Backdoor
- Worm
- Trojan
- Adware

3.3.2.1. Ransomware

The following are the permissions commonly requested by ransomware samples identified in our research, along with detailed descriptions and justifications for their use. This analysis aims to highlight how specific permissions align with the malicious behavior typically exhibited by ransomware.

Permission	Description	Justification
WRITE_EXTERNAL_STORAGE	Allows an application to write to external storage.	This permission allows ransomware to modify or encrypt files on the device's external storage.
READ_EXTERNAL_STORAGE	Allows an application to read from external storage.	used by ransomware to locate and access user files before encrypting them.
READ_PHONE_STATE	Allows read only access to phone state, including	Allows ransomware to access the state of cellular

	the current cellular network information, the status of any ongoing calls, and a list of any PhoneAccounts registered on the device.	network connectivity which might be needed for C2 communications.
KILL_BACKGROUND_PROCESSES	Allows an application to call ActivityManager.killBackgroundProcesses(String)	Used by ransomware, or malware in general, to stop security apps from running in the background.
READ_CONTACTS	Allows an application to read the user's contacts data.	These can be exploited by ransomware, or malware in general, for further spreading of the malware.
SEND_SMS	Allows an application to send SMS messages.	
RECEIVE_SMS	Allows an application to receive SMS messages.	
RECEIVE_BOOT_COMPLETED	Allows an application to receive the Intent.ACTION_BOOT_COMPLETED that is broadcast after the system finishes booting.	This permission allows ransomware, and malicious apps, to restart automatically when the device powers on, ensuring that the ransomware persists even after device reboot.

3.3.2.2. Spyware

The following are the permissions commonly requested by spyware samples identified in our research, accompanied by descriptions and justifications. This analysis outlines how these permissions enable unauthorized surveillance and data exfiltration, which are hallmark behaviors of spyware.

Permission	Description	Justification
READ_SMS	Allows an application to read SMS messages.	Allow malware to steal information from device.
READ_MEDIA_VIDEO	Allows an application to read video files from external storage.	
READ_MEDIA_IMAGES	Allows an application to read image files from external storage.	
READ_MEDIA_AUDIO	Allows an application to read audio files from external storage.	
READ_EXTERNAL_STORAGE	Allows an application to read from external storage.	
READ_CONTACTS	Allows an application to read the user's contacts data.	
READ_CALL_LOG	Allows an application to read the user's call log.	
RECORD_AUDIO	Allows an application to record audio.	
CAMERA	Required to be able to access the camera device.	
INTERNET	Allows applications to open network sockets.	
SEND_SMS	Allows an application to send SMS messages.	Allow malware to send and intercept SMS messages.
RECEIVE_SMS	Allows an application to receive SMS messages.	

3.3.2.3. Worm

As SMS worms are the most prevalent form of mobile worms, we focused our analysis on their permission usage to understand how they facilitate self-replication and spread via messaging. The following are the permissions commonly requested by SMS worm samples identified in our research, along with explanations and justifications. These permissions enable the worm to

propagate via messaging channels and manipulate SMS functionality to spread itself to other devices.

Permission	Description	Justification
READ_CONTACTS	Allows an application to read the user's contacts data.	Allows SMS worms to spread via SMS
SEND_SMS	Allows an application to send SMS messages.	
RECEIVE_SMS	Allows an application to receive SMS messages.	
READ_SMS	Allows an application to read SMS messages.	
INTERNET	Allows applications to open network sockets.	Used for network-based propagation or C2.

3.3.2.4. Backdoor

Backdoors do not exhibit a unique behavioral pattern aside from persistence, and they are often embedded within other malware types such as Trojans or Spyware. As a result, they do not have a distinct permission signature of their own and instead share permission characteristics with their associated malware type.

3.3.2.5. Trojan

Trojans refer to malware that masquerades as legitimate applications to trick users into installing them. The “Trojan” label describes the delivery mechanism, not the specific behavior of the malware post-installation. As a result, Trojans do not have a unique set of permissions associated with them; instead, they inherit the permission patterns of the actual malicious functionality they carry whether that’s spyware, ransomware, or others.

3.3.2.6. Adware

Adware is a type of malware that spams users with unwanted apps [7]. Although less harmful than other malware types, adware can be associated with spyware, leading to the theft of user data. While Android permissions do not explicitly address ad-related activities, adware often requires permissions such as INTERNET and ACCESS_NETWORK_STATE to fetch and display advertisements. If the adware sample is associated with spyware, it is expected to request a subset of the spyware-related permissions.

3.3.3. Raw and Engineered Permission Features

To build a more behaviorally intelligent malware detection system, we designed our feature set around two categories: raw permissions and engineered permission-based features. The raw features consist of 14 Android permissions selected based on our Android malware permission profiling, as demonstrated in the previous section, which highlighted characteristic permission

patterns across different malware types. While these features capture basic capability flags, they lack context about how combinations of permissions might reflect real-world attack patterns. To address this, we engineered 14 high-level features that group related permissions into behavioral patterns commonly associated with malware types like spyware, ransomware, SMS worms, and command-and-control (C2) channels. These engineered features are not only more interpretable but also designed to generalize better across unseen malware samples.
The following tables present both the raw and engineered features used in our model:

Table 3.1 Raw Permission Features

Feature Name	Description
WRITE_EXTERNAL_STORAGE	App can write to external storage (e.g., SD card).
READ_PHONE_STATE	Access phone number, current network info, call state, etc.
KILL_BACKGROUND_PROCESSES	Terminate background processes of other apps.
READ_CONTACTS	Read the user's contact data.
SEND_SMS	Send SMS messages.
RECEIVE_SMS	Receive SMS messages.
RECEIVE_BOOT_COMPLETED	Receive system broadcast when device finishes booting.
READ_EXTERNAL_STORAGE	Read data from external storage.
READ_SMS	Read SMS messages.
READ_CALL_LOG	Read the user's call log.
RECORD_AUDIO	Record audio using the device microphone.
CAMERA	Access the device's camera.
INTERNET	Open network sockets.
ACCESS_NETWORK_STATE	Access info about networks (e.g., Wi-Fi, cellular).
WAKE_LOCK	Prevent the device from sleeping.
ACCESS_LOCATION_EXTRA_COMMANDS	Access extra location provider commands.
ACCESS_WIFI_STATE	View Wi-Fi connection info.

Table 3.2 Engineered Permissions Features

Feature Name	Derived From	Description
persistence_basic	RECEIVE_BOOT_COMPLETED, KILL_BACKGROUND_PROCESSES	Indicates basic persistence capability.
persistence_extended	RECEIVE_BOOT_COMPLETED, KILL_BACKGROUND_PROCESSES, WAKE_LOCK	Extended persistence indicator using wake lock.
c2_sms_exfiltration	SEND_SMS,INTERNET, READ_PHONE_STATE, ACCESS_NETWORK_STATE	Potential SMS-based data exfiltration behavior.

c2_persistence	WAKE_LOCK, INTERNET	C2 infrastructure maintained with network + wake lock.
c2_exfil_with_storage	WAKE_LOCK,INTERNET, READ_EXTERNAL_STORAGE	Indicates possible data exfiltration from storage.
spyware_file_access	CAMERA, ACCESS_LOCATION_EXTRA_COMMANDS	Suggests spying using camera and location access.
spyware_privacy_invasion	CAMERA, RECORD_AUDIO, SEND_SMS	Composite spyware behavior including surveillance and exfiltration.
sms_worm_pattern	SEND_SMS, READ_SMS, READ_CONTACTS	Pattern used by SMS worms to propagate.
ransomware_core	READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE	Minimum requirement for ransomware activity (read/write files).
ransomware_extended	Above + INTERNET, WAKE_LOCK, SEND_SMS	Extended ransomware behavior with exfiltration capabilities.
file_access_count	READ_CALL_LOG, READ_EXTERNAL_STORAGE, READ_CONTACTS, READ_SMS	Number of permissions related to sensitive file access.
network_permission_count	INTERNET, ACCESS_NETWORK_STATE, ACCESS_WIFI_STATE, READ_PHONE_STATE	Number of network-related permissions.
permission_count	All permissions present in the app	Total number of granted permissions (indicator of privilege abuse).
malware_behavior_class_count	Counts non zeros: persistence_extended, c2_sms_exfiltration, spyware_privacy_invasion, sms_worm_pattern, ransomware_extended	How many distinct malware behavior classes a sample exhibits.

3.4. Neural Networks

Artificial neural networks (ANNs) are computational models inspired by the structure and functioning of the human brain. They consist of interconnected layers of simple processing units called neurons, which collectively learn patterns from data through weighted connections. Neural networks are particularly effective for tasks involving complex, non-linear relationships, such as image classification, natural language processing, and, in this context, malware detection. Their

ability to automatically extract features and generalize from large datasets makes them well-suited for identifying subtle and obfuscated malicious behaviors in Android applications.

In this project, we developed a neural network for binary classification using 28 input features. The architecture comprises two hidden layers with 64 and 32 neurons, respectively, both using the ReLU activation function to introduce non-linearity. The output layer consists of a single neuron with a sigmoid activation function, producing a probability score to classify applications as either benign or malicious. The model was implemented using TensorFlow, which by default handles backpropagation during training to optimize the network's weights.

The following diagram illustrates the architecture of the neural network used in this project.

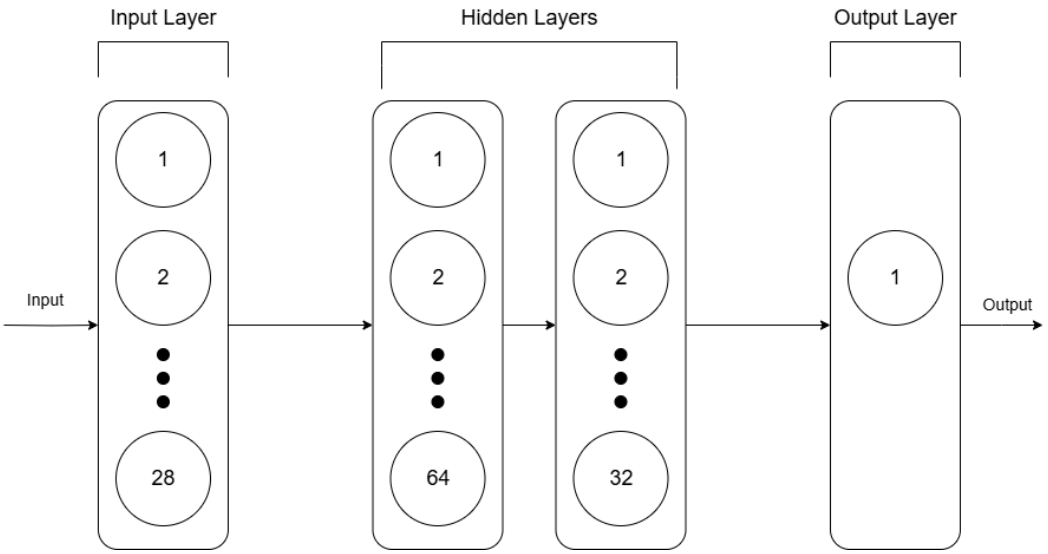

Figure 3.1 Neural Networks Architecture

3.5. Generalization Tests

To evaluate the effectiveness and generalization capability of our malware detection system, we conducted a series of experiments using different Android malware datasets across various time periods. Our primary goal was not only to achieve high accuracy on training data but also to assess how well the model generalizes to unseen and evolving threats.

We performed several trials where each dataset was used for training, and different datasets were used for testing. This allowed us to observe the generalization behavior of the model when exposed to malware samples from different time frames and characteristics. The results highlight how training on outdated data may fail to capture modern malware patterns, and how combining datasets can improve both robustness and adaptability.

The following table summarizes the performance of our model in terms of test accuracy and malware recall across these trials.

Table 3.3. Generalization Tests

Training Dataset	Test Accuracy	Malware Recall	Generalization Dataset	Generalization Accuracy	Malware Recall
TUANDROMD	95%	96%	COLCOM	61%	27%
TUANDROMD	95%	96%	Drebin (subset)	72%	38%
Drebin (subset)	90%	88%	COLCOM	94%	93%
Drebin (subset)	90%	88%	TUANDROMD	40%	27%
Hybrid (Drebin + TUANDROD)	91%	84%	COLCOM	90%	82%

While some datasets, such as TUANDROMD, achieved higher accuracy during training and testing (e.g., 95% test accuracy and 96% malware recall), they failed to generalize well to other datasets, with generalization accuracy dropping to as low as 61% and malware recall plummeting to 27% on COLCOM. Similarly, Drebin-trained models generalized poorly to TUANDROMD (40% accuracy, 27% recall) despite performing well on COLCOM. On the other hand, the hybrid dataset—a combination of Drebin and TUANDROMD—achieved a balanced performance, maintaining both respectable training accuracy (90%) and strong generalization results (91% accuracy, 84% malware recall on COLCOM). This balance between robust generalization and solid in-distribution performance made the hybrid dataset the most suitable choice for training our neural network and deploying it within our Android application.

3.6. Model Evaluation

As highlighted earlier, our neural network was trained on a hybrid dataset combining TUANDROMD, which represents newer malware, and DREBIN, capturing older malware samples. The figure below illustrates the distribution of malware and benign samples within this dataset—showing a nearly balanced split, which effectively eliminates any class imbalance concerns.

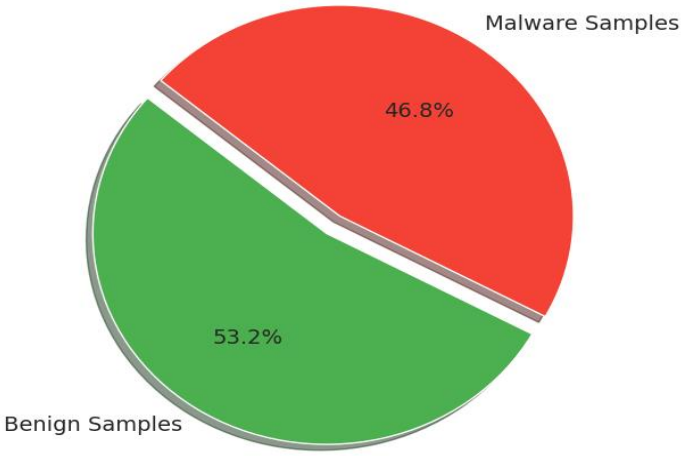

Figure 3.2 Distribution of Benign and Malware Samples in our Hybrid Dataset

The model was trained over 27 epochs. The following figures display the loss and AUC metrics throughout training. We observe that the loss steadily decreases and nearly stabilizes by the end, indicating effective learning, while the AUC gradually increases and also levels off, reflecting improved and consistent performance.

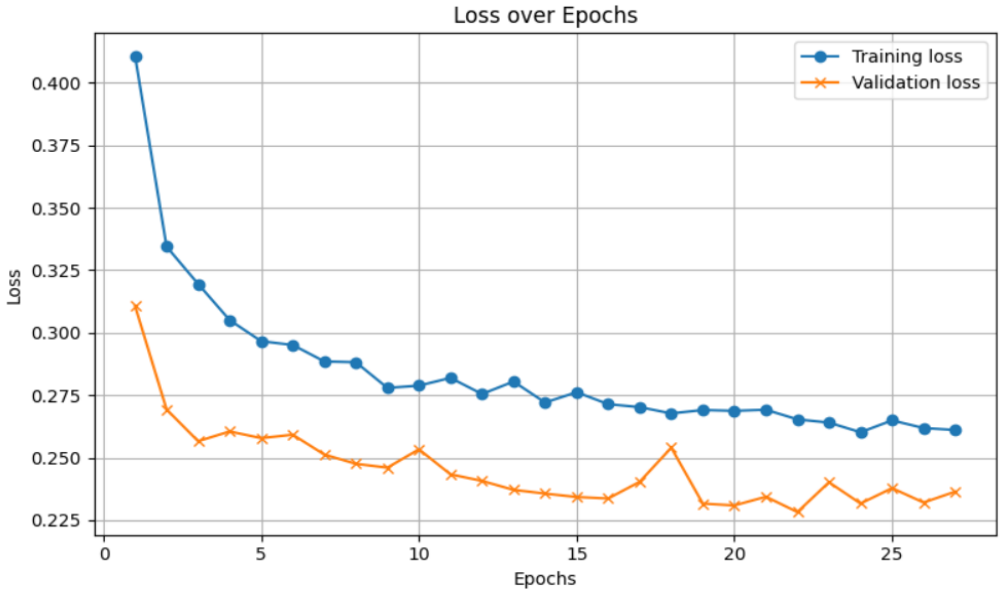

Figure 3.3 Loss over Epochs

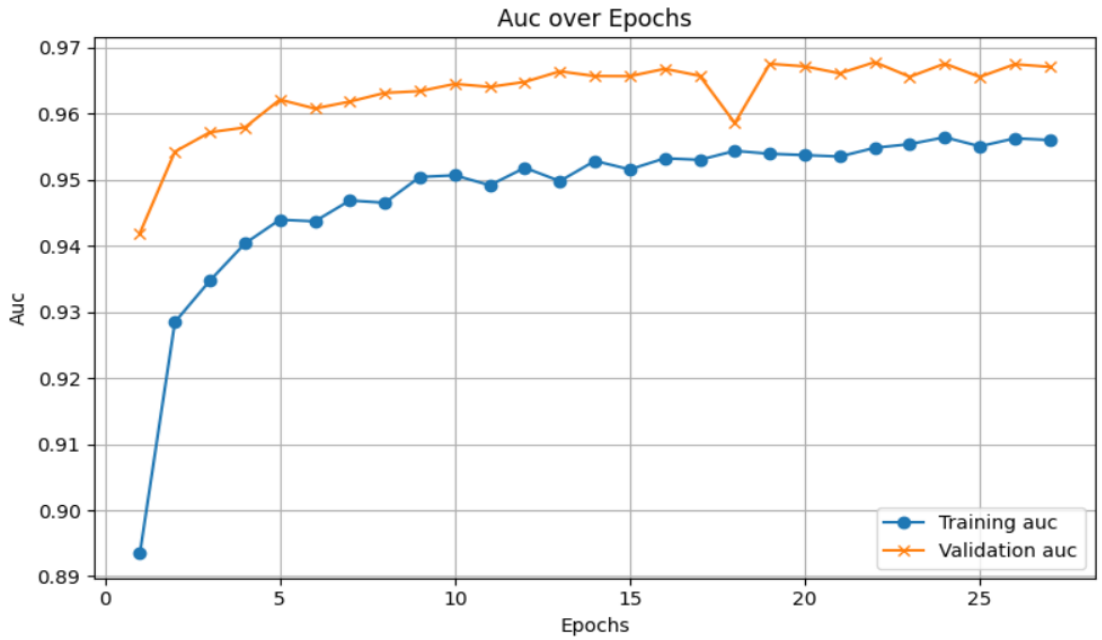

Figure 3.4 AUC over Epochs

The following figure presents the confusion matrix for the test results. Out of 2087 goodware samples, 1973 were correctly classified while 114 were misclassified as malware. For the 1813 malware samples, 1538 were correctly detected, with 275 mistakenly classified as goodware.

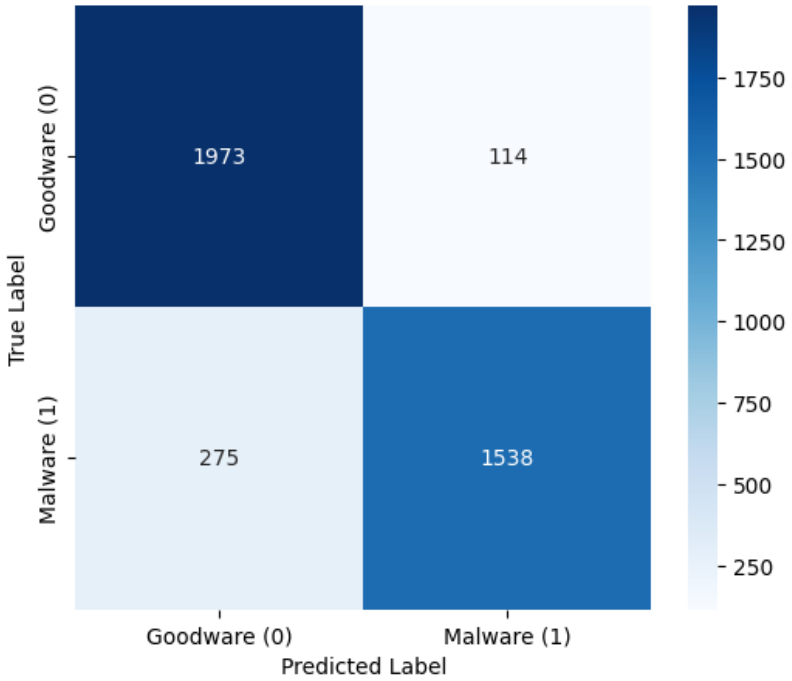

Figure 3.5 Confusion Matrix

Overall, the model demonstrates a reasonably low rate of false positives and false negatives, reflecting solid classification performance.

The following figure visualizes the classification performance of the malware detection model. It shows precision, recall, and F1-score for both classes: Goodware and Malware. The model achieved high recall for Goodware (0.95), meaning it correctly identified most benign apps, while its recall for Malware (0.85) indicates it missed some malicious samples. Precision is slightly higher for Malware (0.93), showing that most predicted malware samples were indeed malicious. The overall macro and weighted averages for all metrics are balanced at 0.90, reflecting consistent performance across both classes.

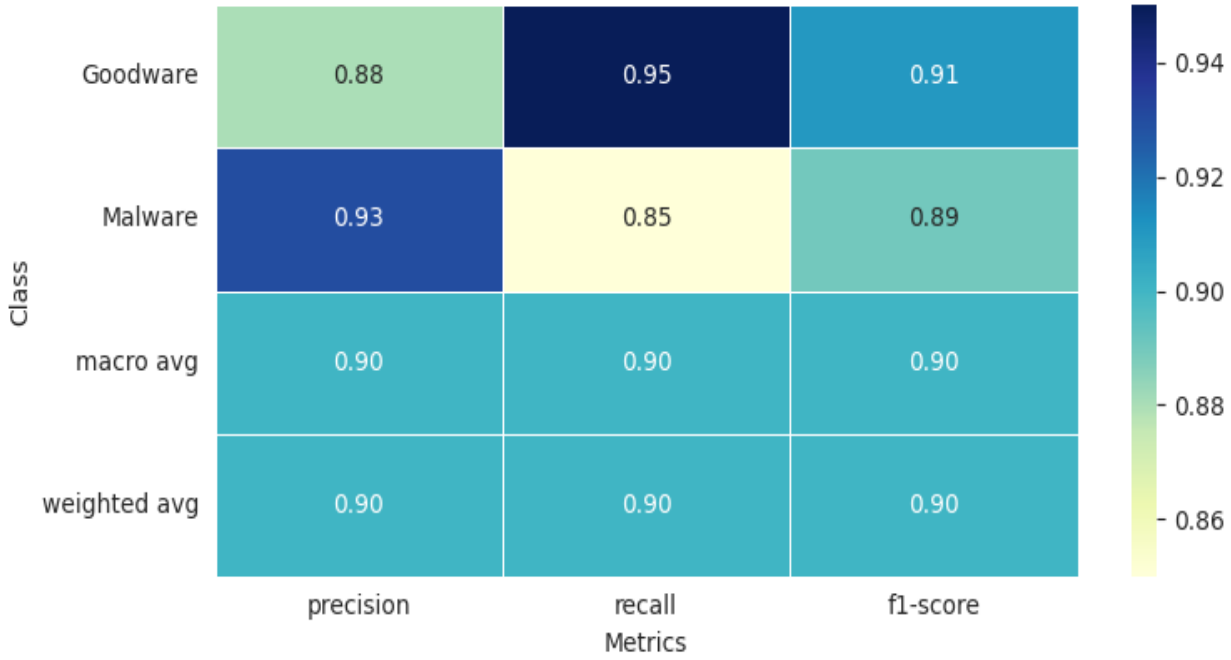

Figure 3.6 Classification Report Heatmap

We then conducted generalization testing on the COLCOM dataset to assess the model's performance beyond the training data, with a focus on minimizing false negatives. The model achieved an overall accuracy of 91%, with the benign class (0) showing a precision of 0.86 and a high recall of 0.98, while the malware class (1) had a precision of 0.98 and a recall of 0.84.

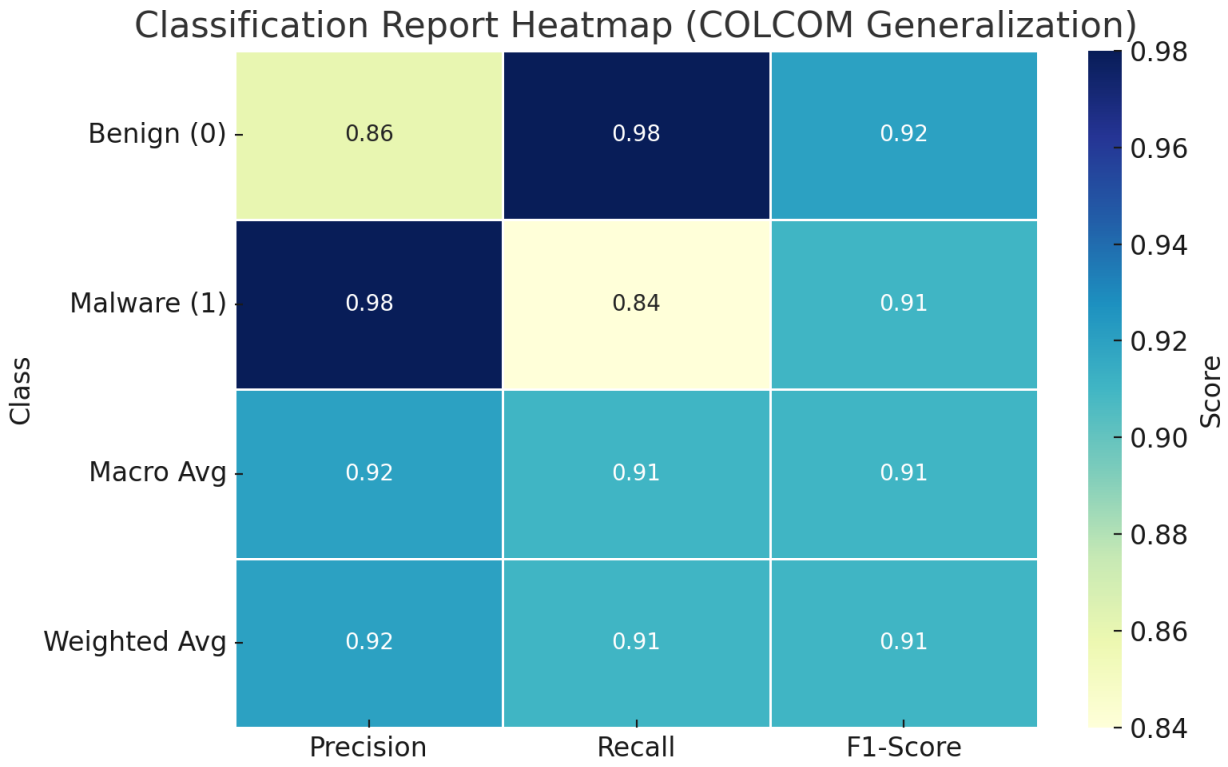

Figure 3.7 Generalization Tests

The following confusion matrix indicates 3 false positives and 32 false negatives, demonstrating that the model maintains a strong ability to correctly identify benign samples and keeps false alarms low, although some malware instances were missed. This outcome highlights a solid balance between detection and false negative reduction during generalization.

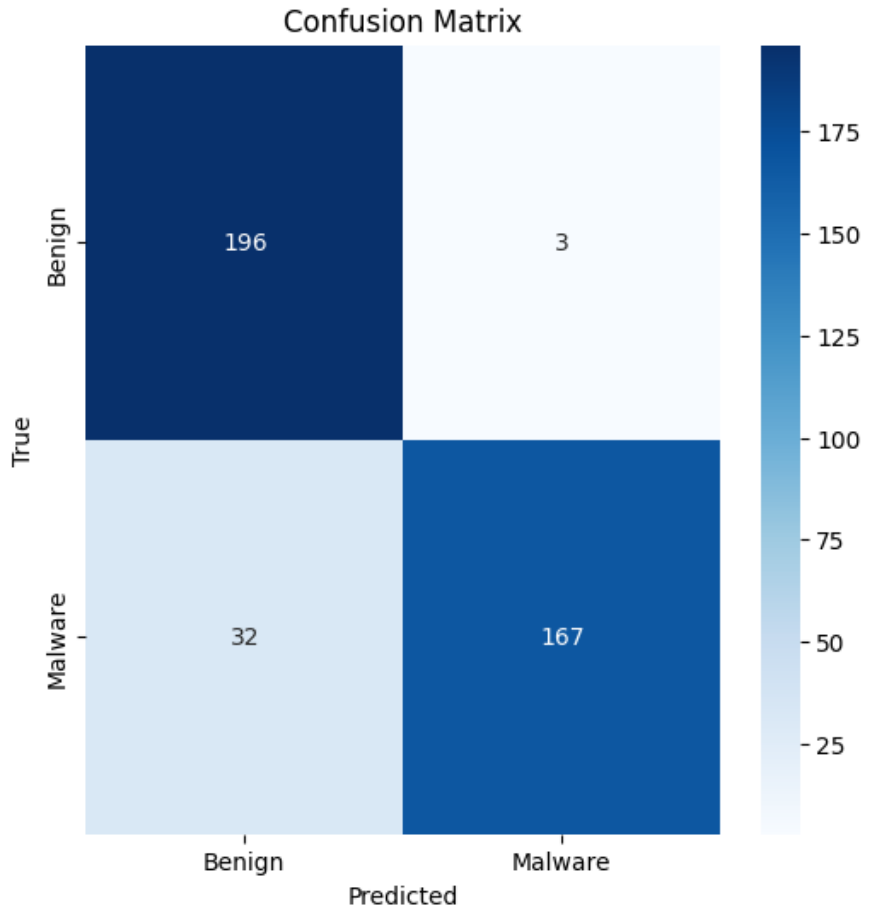

Figure 3.8 Generalization Confusion Matrix

3.7. Large Language Model (LLM)

3.7.1. LLM Overview

This section introduces what LLMs are and sets the stage for deeper technical explanation. Large Language Models (LLMs) are advanced AI systems trained on massive amounts of textual data to understand and generate human-like language. These models, such as OpenAI's GPT series, are based on transformer architecture and have demonstrated impressive capabilities across a range of tasks like translation, summarization, question answering, and code generation.

3.7.2. OpenRouter.ai

To implement the chatbot feature in our Android application, we utilized the API provided by OpenRouter.ai, a powerful gateway that connects developers to a wide variety of large language models (LLMs) through a unified and flexible interface. OpenRouter.ai simplifies integration by acting as a bridge between client applications and multiple LLM providers, including open-source and commercial models. It supports modern authentication, rate limiting, and efficient routing, which allows for reliable and scalable access to advanced AI capabilities without hosting the models locally. This flexibility made it an ideal choice for my mobile app, enabling me to select the best-fit model based on performance and resource requirements. we selected Mistral: Mistral 8B, an open-weight language model known for its efficiency and high performance in conversational AI tasks. Mistral 8B is a transformer-based model with 8 billion parameters, optimized for fast inference and strong reasoning capabilities. It is particularly well-suited for mobile and embedded applications due to its balance between performance and computational cost.

3.7.2.1. Comparison of Language Models on OpenRouter.ai

To evaluate the most suitable model for Patronus, we compared Mistral: Mistral 8B with some of the most popular models available on OpenRouter.ai. The table below highlights key differences in terms of model architecture, performance, cost, and suitability for mobile chatbot integration.

Table 3.4 Comparison of Language Models on OpenRouter.ai

Model	Developer	Parameters	Open Source	Speed	Accuracy	Cost Efficiency	Best For
Mistral 8B	Mistral AI	8B	Yes	Fast	Moderate to High	Very Affordable	Mobile chatbots, real-time apps
GPT-4	OpenAI	~1T (MoE)	No	Medium-High	Very High	Expensive	Enterprise AI, deep reasoning
Claude 3 Opus	Anthropic	~200B (est.)	No	Medium-High	Very High	Expensive	Long-context reasoning, AI agents
Gemini 1.5 Pro	Google	~300B (est.)	No	Medium	Very High	Expensive	General-purpose AI, long-

							context tasks
LLaMA 3 (8B/70B)	Meta	8B / 70B	Yes	Fast	High	Affordable	Chatbots, research, fine-tuning

Among the various models available on OpenRouter.ai, Mistral: Mistral 8B stands out as the most suitable choice for lightweight, real-time applications such as mobile chatbots. Compared to larger proprietary models like GPT-4 (OpenAI), Claude 3 Opus (Anthropic), and Gemini Pro (Google), Mistral 8B offers significantly faster response times and lower operational costs. While it may not reach the same level of raw intelligence, contextual depth, or long-form reasoning as these heavyweight models, its open-source nature, efficient performance, and cost-effectiveness make it ideal for mobile environments where speed, responsiveness, and resource constraints are key considerations. This balance between capability and efficiency is precisely why Mistral 8B was selected for the chatbot

3.7.2.2. Advantages and Disadvantages of Mistral 8B

The following table summarizes the key advantages and limitations of integrating the Mistral 8B language model via OpenRouter.ai in our Android application. We considered aspects that directly affect chatbot performance and user experience, including accuracy, integration flexibility, scalability, cost, and ethical considerations. This structured overview highlights why Mistral 8B was chosen for our chatbot feature—offering efficient performance and seamless integration—while also acknowledging limitations such as potential hallucinations and costs associated with API-based access. These factors were critical in ensuring the chatbot meets our technical and ethical standards.

Table 3.5 Advantages and Disadvantages of Mistral 8B

Aspect	Advantages	Limitations
Accuracy & Performance	• Fast, efficient inference for chatbot tasks.	• Can still generate hallucinated or misleading responses.

	• Well-suited for mobile apps due to optimized architecture.	
Integration & Flexibility	• Easy API-based integration via OpenRouter.ai. • Choice of LLMs for different needs (open-source and commercial).	• Limited direct control over model fine-tuning or weights (requires more resources if needed).
Scalability & Cost	• Scalable access without hosting models locally. • Mistral's open-weight licensing can reduce licensing costs.	• API usage costs can increase at scale.
Ethical Considerations	• Uses reputable open-source model with transparent licensing.	• Potential biases or misuse if not monitored responsibly.

3.8. Conclusion

This chapter provided a detailed technical foundation for machine learning-driven Android malware detection. We explored key datasets and outlined the feature engineering strategies critical for identifying malicious applications based on permission profiles. Neural network models were evaluated for their detection performance, and generalization tests highlighted their robustness across datasets. The inclusion of LLMs represents a promising direction for enhancing intelligent analysis. These components collectively prepare the groundwork for building the system's functional architecture, as detailed in the next chapter.

Chapter 4

System Design and Techniques

4.1. Introduction

This chapter describes the architecture and implementation of the proposed Android security application. It outlines the major components of the system, including network scanning, AI-powered malware detection, security modes, HelpBot, real-time system monitoring, and threat remediation. Each module's workflow is discussed in detail, supported by component and use case diagrams that demonstrate the interactions among system elements. The user interface design is also presented to illustrate the end-user experience across different functional screens.

4.2. Overview of Application Components

The proposed Android application is designed with a modular architecture that combines real-time threat detection, intelligent automation, and user-centric security controls. The core components are as follows: The **Network Scanning Module** includes both a Wi-Fi Scanner that analyzes public networks for potential vulnerabilities (e.g., speed, type, and encryption status) and an **App-Level Network Scanner** that monitors active connections by installed apps, comparing their destination IPs against a built-in **IP Reputation Database** to detect suspicious communications. The **AI-Powered Malware Detection Module** analyzes newly installed applications to identify potentially harmful behavior using machine learning techniques. Upon detection of malicious activity, the **Threat Remediation Component** takes appropriate action such as blocking the app or alerting the user. Users interact with the system via a dynamic **Notification System** and an integrated **AI-powered Chatbot**, built using OpenRouter.ai API, which helps explain threats and guides user response in natural language. The app also features **Real-Time System Monitoring** to observe device behavior and app installations, based on selected **Security Modes**—Manual (user-controlled), Smart (adaptive rule-based). Together, these components form a comprehensive, intelligent mobile defense system that addresses both app-based and network-based threats with minimal user friction.

4.3. Component Diagram

This component diagram illustrates Patronus architecture. At the core of the system is the **Android System**, which interacts with various modules to ensure real-time monitoring and protection against network and malware threats. The **Network Scanning Module**, comprising a Wi-Fi Scanner and App-Level Network Scanner, collects network data and utilizes the **IP Reputation Database** to identify potential network threats. Detected threats are sent to the

Threat Remediation Component for appropriate action. The **AI-Powered Malware Detection Module**, which uses the **Android Malware Detection Neural Network Model**, receives data on installed applications to identify malware and forwards threats to the remediation component. Additionally, a **Real-Time System Monitoring** module listens to package installation broadcasts to detect changes in the system, while the **Security Mode Selector**—with Smart and Manual selector modules—adjusts security settings based on user preference or system intelligence. Notifications about threats and system status are managed through the **Notification System Component**, which also interfaces with a **Chat Bot** that leverages the **OpenRouter.ai (Mistral) API** for user interaction. This modular design ensures layered security, flexibility, and responsiveness in maintaining Android device integrity.

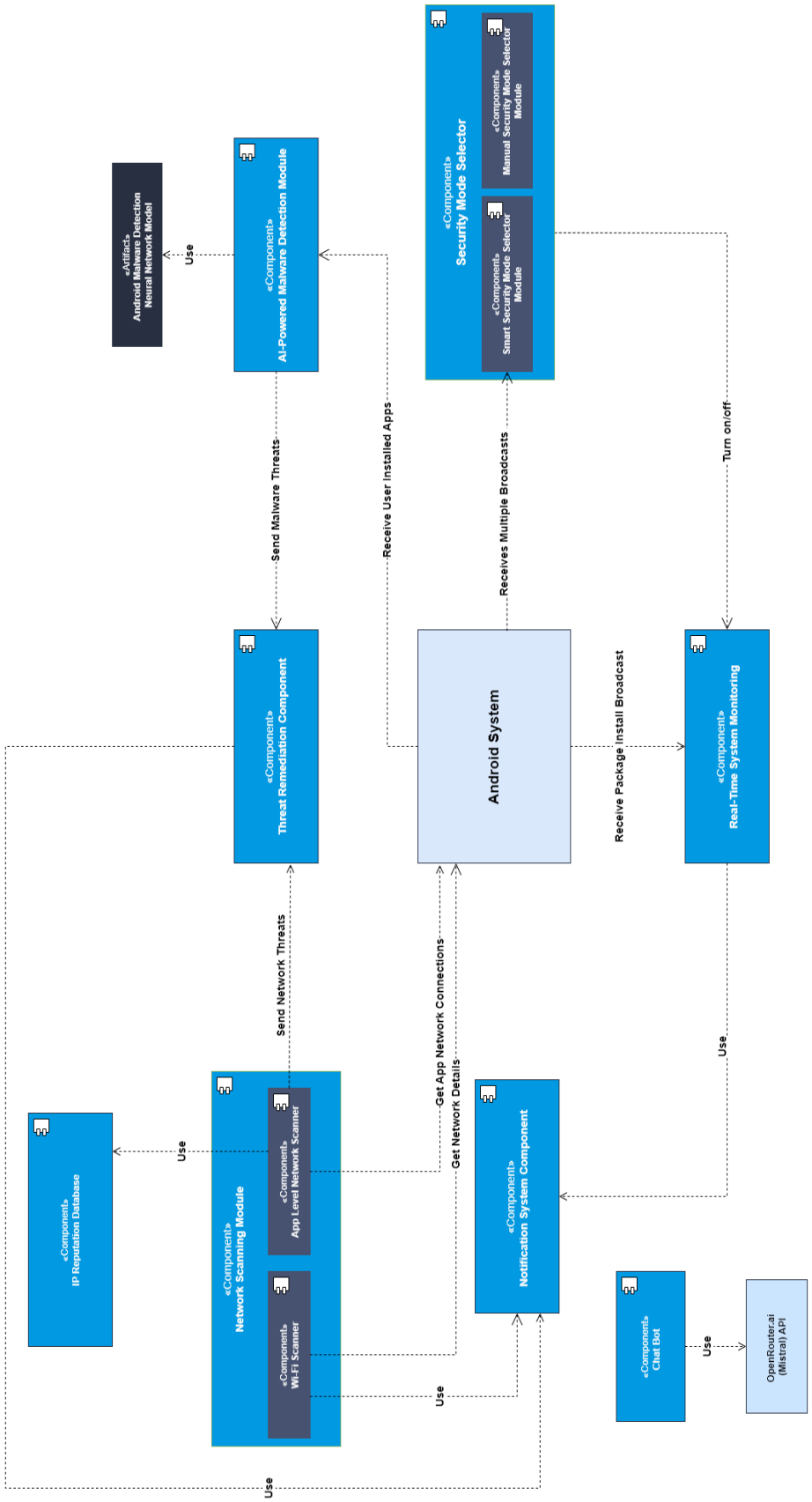

Figure 4.1 Component Diagram

4.4. Detailed component workflows

4.4.1. Network Scanning Module

4.4.1.1. Wi-Fi Scanner Component

4.4.1.1.1. Wi-Fi Threat Detection and Risk Assessment

This activity diagram represents the workflow of the Wi-Fi Threat Detection and Risk Assessment feature. The process begins when the app is launched, at this point it checks whether location permissions have been granted; if not, it prompts the user to grant them. It then verifies if Wi-Fi is enabled on the device—if not, it prompts the user to enable it. Once Wi-Fi is active, the app retrieves the current network's SSID, BSSID, and RSSI, and then determines the encryption type. A Wi-Fi scan is initiated to detect nearby networks. For each scanned network, the app calculates a risk score to evaluate the threat level. Based on the score, the app categorizes the network as high risk (score ≥ 5), moderate risk (score ≥ 3 but < 5), or low risk (score < 3). Depending on the assessment, it displays the appropriate threat level and provides notification suggestions to the user. This structured approach enables real-time identification and communication of potential Wi-Fi security threats.

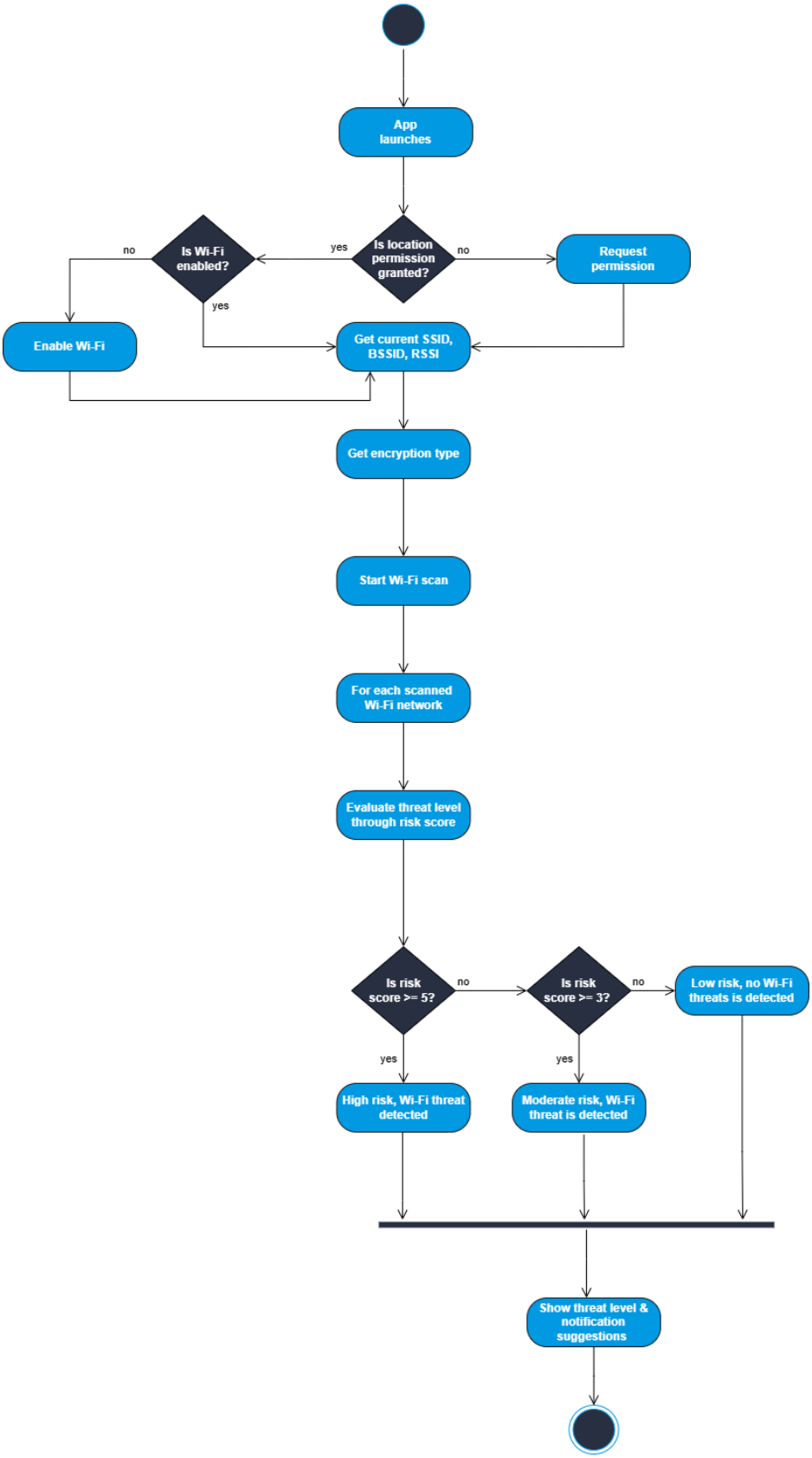

Figure 4.2 Wi-Fi Threat Detection & Risk Assessment Activity Flow

4.4.1.1.2. Wi-Fi Scan & Diagnostic Logic (Network Center)

This activity diagram represents the workflow of the “Network Center” feature, which serves as a centralized hub for users to assess and interact with key elements of their current Wi-Fi connection. This feature consists of three primary modules: General Info, Speed Test, and Troubleshoot. The General Info module gathers and displays essential network parameters such as SSID, BSSID, IP address, network type, and security protocol. It also conducts a basic check for rogue access points by analyzing inconsistencies in the SSID. The Speed Test module evaluates network performance by measuring real-time upload and download speeds through HTTP connections to predefined endpoints. The speed values are then compared against predefined thresholds to classify the connection quality as Excellent, Moderate, or Poor. For example, connections with high transfer rates and low latency are marked as Excellent, while lower bandwidth and higher delays trigger Moderate or Poor classifications. Based on these results, the system provides tailored suggestions to help the user enhance performance, such as reconnecting to the network or checking signal strength. The Troubleshoot module diagnoses connectivity issues by attempting to establish a connection with a reliable DNS server (8.8.8.8) and executing ping tests to detect potential interruptions or latency spikes along the network path. All test outcomes are presented through intuitive alert dialogs for clarity. Additionally, the activity handles runtime permission requests—particularly for location access—to ensure the retrieval of accurate network data while safeguarding user privacy.

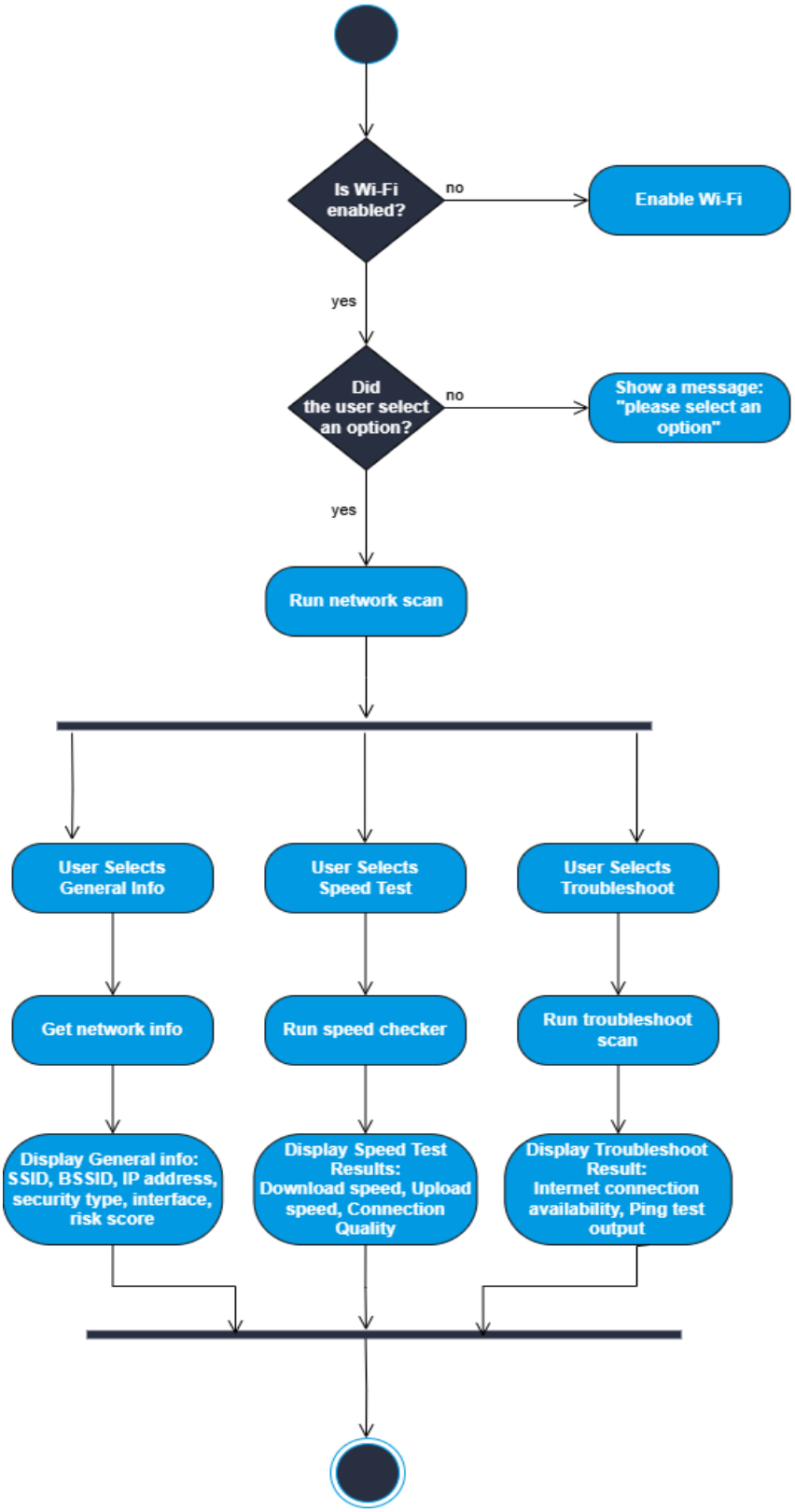

Figure 4.3 Network Center Activity Flow Diagram

4.4.1.2. Application-Level Network Scanner

This activity diagram illustrates the process of the "Application-Level Network Scanner" feature which is designed to detect malicious IP communication on the application level. The workflow begins by checking whether network monitoring is enabled. If it is not, the system loops back and continues to check. Once monitoring is active, the app retrieves a list of all installed applications. It then gathers TCP and UDP connection data for each app from system files (/proc/net/tcp and /proc/net/udp). The remote IP addresses from these connections are extracted and parsed. Each extracted IP address is scanned against a local threat intelligence database to identify any potentially malicious activity. If a malicious IP is found, the user is notified and presented with appropriate remediation actions. Regardless of the result, the system enters a 5-minute wait period before the monitoring cycle restarts, ensuring periodic and continuous threat assessment. This feature is only available on Android versions below 10, as access to the /proc/net/ directory—which is essential for scanning remote connections—is restricted by the security model introduced in Android 10. However, it has been tested and functions reliably on devices running Android 9 and earlier.

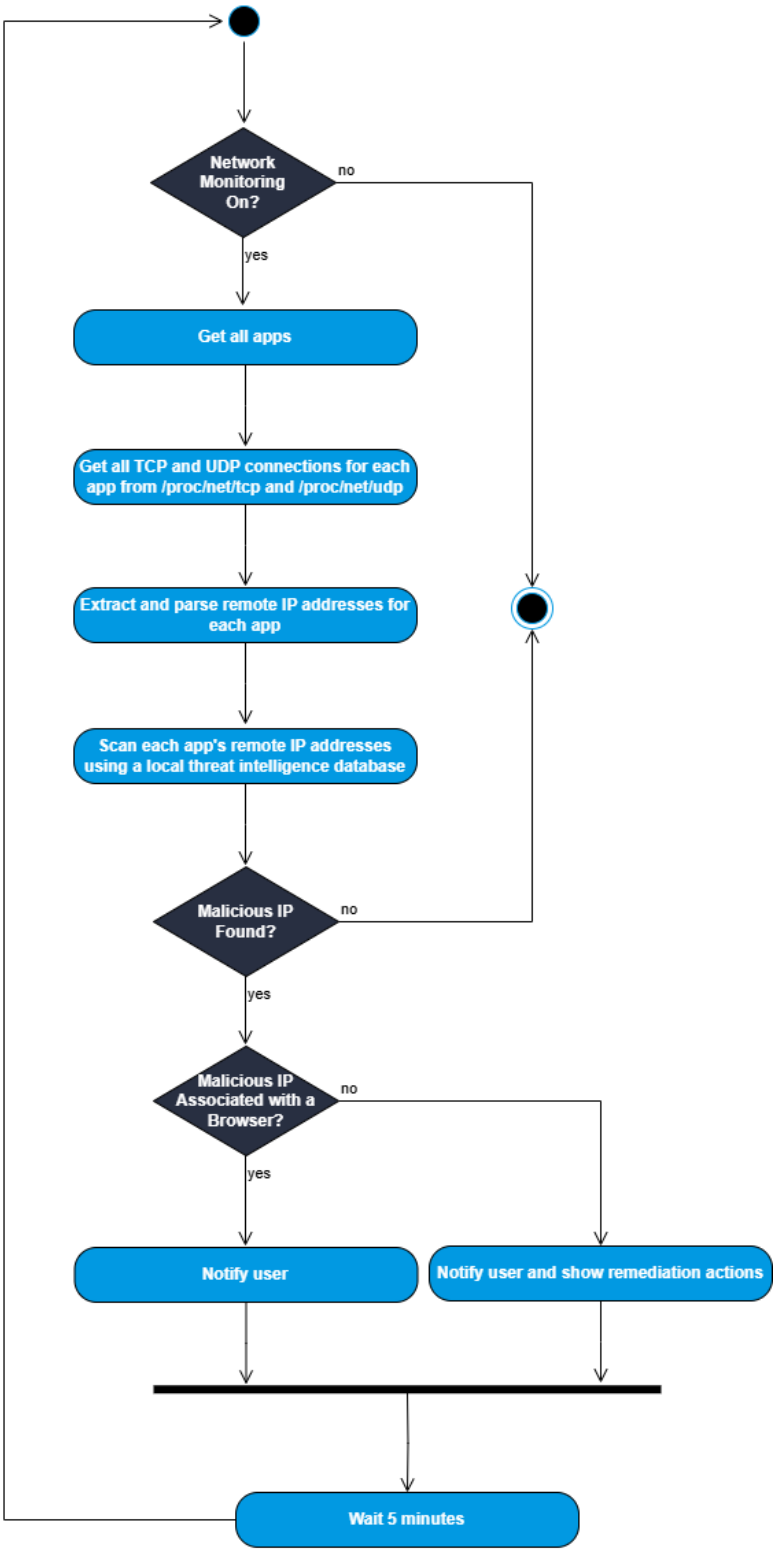

Figure 4.4 Application-Level Network Scanner Activity Flow Diagram

4.4.1.2.1. Safe Browsing

Our application-level network threat detection inherently provides safe browsing functionality, as it monitors suspicious network connections initiated by individual applications — including browsers, which are essentially apps that establish connections with web servers. However, our standard remediation action of uninstalling the offending app is not suitable in the case of browsers. It would be unreasonable to recommend deleting a browser solely because the user visited a website hosted on a server with a suspicious IP address.

For this reason, in cases involving browsers, the system does not suggest removal. Instead, it issues a warning to the user, aligning with the approach commonly adopted by other Android security solutions — including those discussed in the similar software section of Chapter 2 — which alert users but refrain from taking direct action.

Additionally, as seen in solutions like Bitdefender Security, the safe browsing feature is limited to a predefined list of supported browsers. Browsers outside this list are treated as regular applications and are subject to the same remediation measures. The selected browsers — Google, Chrome, Firefox, Opera, and Brave — were chosen based on their popularity and serve as a proof of concept. This list can be extended in future iterations of the system.

Figure 4.5 Browsers Supported by the Safe Browsing Feature

4.4.2. AI-powered Malware Detection Module

The activity flow diagram illustrates the process of detecting malicious applications using an AI-powered system, divided into two main components: the **User Interface** and the **Malware Detection Engine**. The process begins when the user specifies a scan buffer, which defines the scope of the apps to be analyzed. The user is then prompted to either scan *"All Apps"* or manually select specific ones. If the user chooses to scan all apps, the system utilizes the **Package Manager**

API to automatically select all user-installed apps. Otherwise, the system displays a list of installed apps for manual selection.

Once the selection is made, the flow proceeds to the **Malware Detection Engine**, where the selected apps are processed. Initially, the scan buffer is set to the chosen apps. The system then extracts each app's permissions and converts them into a feature vector. These vectors are passed through a **pre-trained neural network model** that calculates a confidence score indicating the likelihood of each app being malicious. To improve accuracy, a **context-aware score** is then computed for each app, incorporating factors such as app category risk, sideloading status, and the neural model's confidence. Finally, the system displays a list of potentially malicious apps along with suggested remediation actions, completing the analysis cycle.

The Context Scoring System enhances detection accuracy by significantly reducing false positives, which are likely to occur if classification relies solely on the neural network. This is because no training dataset can fully capture the wide range of permission patterns found in legitimate applications—particularly those in high-permission categories such as social media or gaming, where elevated permission usage is often justified by functionality. To address this limitation, we augment the model's raw confidence score with additional contextual factors: a category risk score and a sideloading score. The following formula is used by our application to compute the context-aware score:

0.5 × Model Confidence + 0.3 × Category Risk + 0.2 × Sideloading Score

Category risks are assigned based on the table below, which defines predefined risk values for each category according to their prevalence in malware distribution. Categories such as games and accessibility receive some of the highest risk scores, reflecting their frequent use as vectors for malware. This is especially true because these categories lend themselves well to social engineering attacks, as users are generally more easily persuaded to download a game than a productivity application.

Table 4.1 Category Risk Scores

Category Code	Category Description	Risk Value
-1	Undefined	0.8
0	Games	0.6
1	Audio/Music apps (e.g., music players)	0.2
2	Video/Movies apps (e.g., streaming apps)	0.4
3	Images/Photos apps (e.g., camera or gallery apps)	0.1

4	Social/Communication apps (e.g., messaging, email, social networks)	0.2
5	News apps (e.g., newspapers, magazines, sports apps)	0.3
6	Maps apps (e.g., navigation apps)	0.4
7	Productivity apps (e.g., cloud storage, workplace tools)	0.3
8	Accessibility apps (e.g., screen readers)	0.7
default	Fallback for any category not listed above	0.5

Sideloaded applications present a significant security risk compared to apps distributed through the Google Play Store, which undergo mandatory scanning before release. Since sideloaded apps are not subjected to such rigorous scrutiny, they inherently carry higher risk. To account for this, our system assigns a sideloading score of +1 to sideloaded apps, increasing their overall risk score. Conversely, apps installed from official stores receive a sideloading score of -1, which reduces the overall score by 20%.

The table below presents the performance results of the proposed context-aware malware detection system when deployed on a physical Android device. The test device operated on Android version 8.0.0 and featured a Qualcomm Snapdragon 430 processor, 3 GB of RAM, and 32 GB of internal storage. The scan time recorded for this scan was **132 milliseconds**, indicating the system's high efficiency.

Table 4.2 Context-Aware Scoring Performance Results on a Physical Android Device

App	Category	Permissions	Sideloading status	Context-Aware Score	Label
Personalized Music Suggester App	-1 (undefined)	Light	Sideloaded	0.47 (benign)	benign
Facebook, Whatsapp, Telegram, Instagram	4 (social)	Heavy	Not Sideloaded	0.36 (benign)	benign
Candy Crush Saga	0 (game)	Moderate	Not Sideloaded	0.48 (benign)	benign

The following table summarizes the performance of the proposed context-aware malware detection system when deployed on an Android emulator. The emulator was running Android version 8.1.0 with 1536 MB (1.5 GB) of allocated RAM, reflecting the memory constraints of typical mid-range physical Android devices to ensure a realistic testing environment. The scan time recorded for this scan was **47 milliseconds**.

Table 4.3 Context-Aware Scoring Performance Results on an Android Emulator

App	Category	Permissions	Sideloading status	Context-Aware Score	Label
Bitdefender Mobile Security, Nord VPN	7 (productivity)	Heavy	Not Sideloaded	0.39 (benign)	benign
Spotify	1 (music/audio player)	Heavy	Not Sideloaded	0.36 (benign)	benign
Bottle Shoot (Modified for C2 Simulation)	-1 (undefined)	Heavy	Sideloaded	0.94 (malicious)	malicious

Please note that the used version of the casual game “Bottle Shoot” was intentionally modified to include a reverse shell payload, simulating command and control (C2) communication with a remote attacker. The modification was performed for controlled testing of the malware detection system’s ability to identify stealthy, real-world threats embedded in benign-looking applications. This does not reflect the original application available publicly.

The results from the preceding two scans highlight the system’s effectiveness in accurately predicting application labels. Notably, no false positives or false negatives were observed across the evaluated test cases.

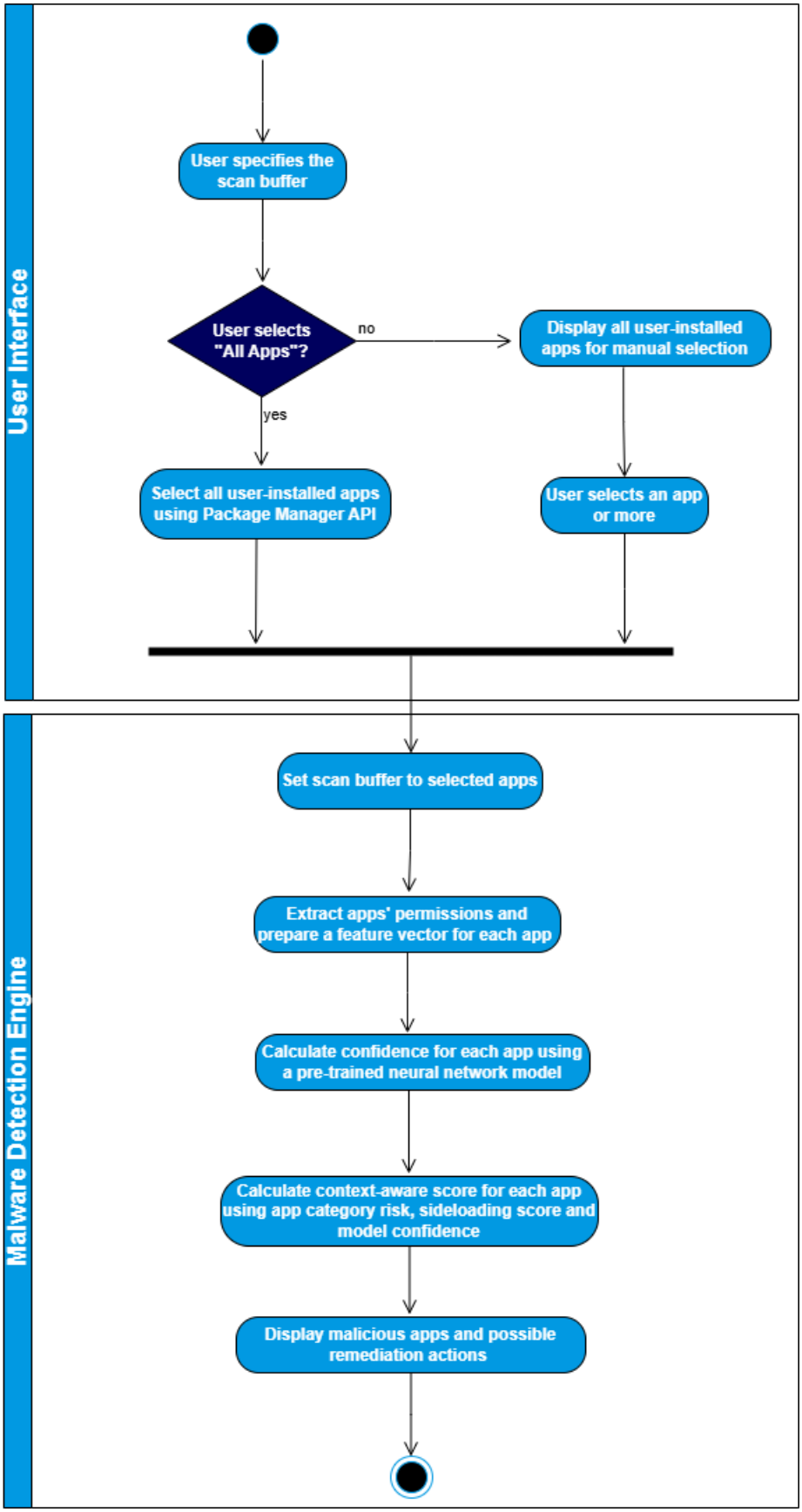

Figure 4.6 AI-powered Malware Detection Activity Flow Diagram

4.4.3. Security Modes Module

This activity flow diagram outlines the logic for managing security modes based on user settings and system status, divided into **user interaction** and **automatic monitoring behavior**. The process begins when the user opens the settings screen and selects a mode via a switch. If the user selects the **"High"** mode, the system immediately starts monitoring security-related activities and maintains this mode without further conditional checks.

If the user instead selects the **"Balanced"** mode, the system performs continuous checks on three key factors: **battery level**, **charging mode**, and **power-saving status**. If the device is either charging or has a battery level above 50%, the system then evaluates whether **power saving is turned off**. Only if both conditions are met—charging (or battery >50%) and power saving is off—the system activates **high security monitoring**.

If either condition fails (i.e., not charging/low battery or power saving is on), the system halts monitoring to conserve resources. The loop then continues, rechecking system status to determine if conditions have changed and if monitoring can be resumed.

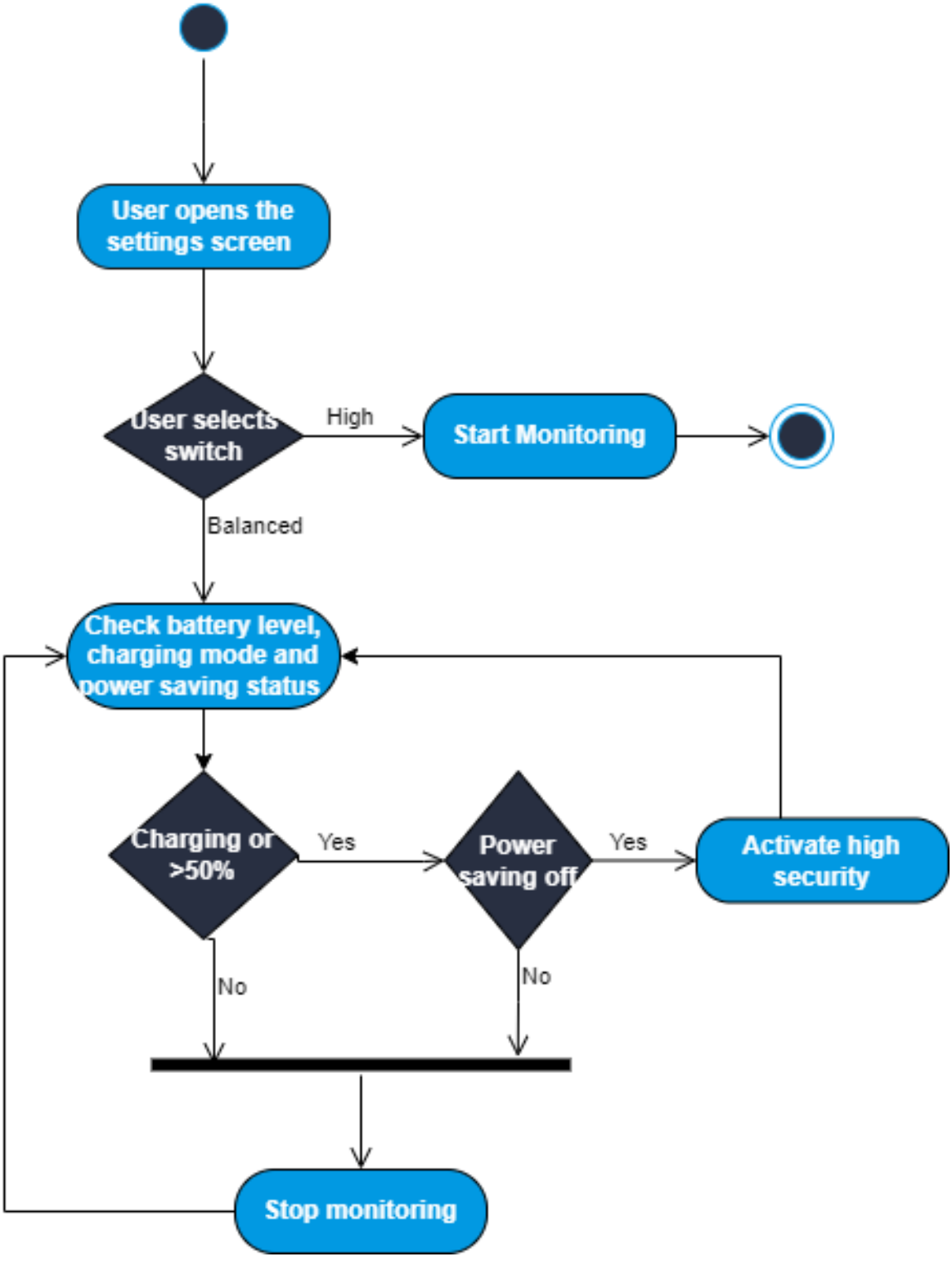

Figure 4.7 Security Modes Activity Flow Diagram

4.4.4. HelpBot Module

This activity flow diagram illustrates the operational process of a **Help Bot** using a large language model (LLM) through an API. The interaction begins when the **user enters a message**. The system first checks if the input message is empty. If it is, the flow is terminated. Otherwise, the message is **displayed in the user interface (UI)**, initiating a call to the **QueryLLM** function.

The next steps involve backend preparation: the system **builds a JSON request** embedding the user's input and **sends an HTTP POST request** to the **OpenRouter API**. After sending, it **waits for a response** from the API. Once a response is received, the system checks if the response is **valid**. If it is not valid, an **error message** is shown to the user, ending the flow. If the response is valid, the system **extracts the bot's message from the JSON structure** and **displays the response** in the UI, completing the interaction loop.

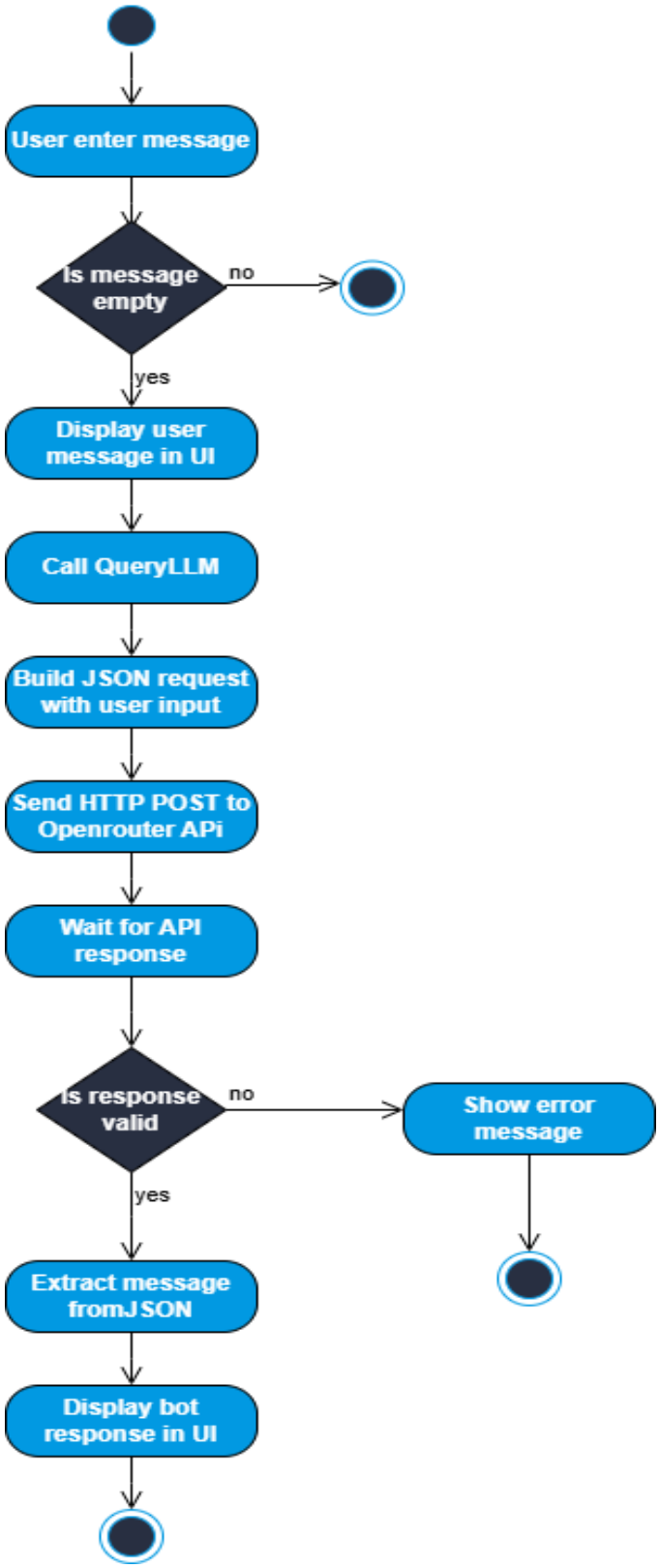

Figure 4.8 Help Bot Activity Flow Diagram

4.4.5. Real-time System Monitoring

This activity flow diagram represents a structured decision-making process for managing three distinct monitoring services: balanced mode, malware monitoring, and network monitoring. It is executed every time the user changes the settings, ensuring that the system responds dynamically to configuration updates. The flow begins by checking whether balanced mode is activated and running; if it is not, the system is prompted to turn it on. The same verification and activation steps are then applied to malware monitoring, followed by network monitoring. Each monitoring service is individually assessed to confirm its status and is activated if found to be inactive or off. After confirming that all three services are appropriately configured and operational, the flow evaluates whether all settings have been turned off. If this condition is met, the system proceeds to stop the network monitoring background service and concludes the workflow. This logic ensures that critical monitoring services are consistently aligned with user preferences and system requirements.

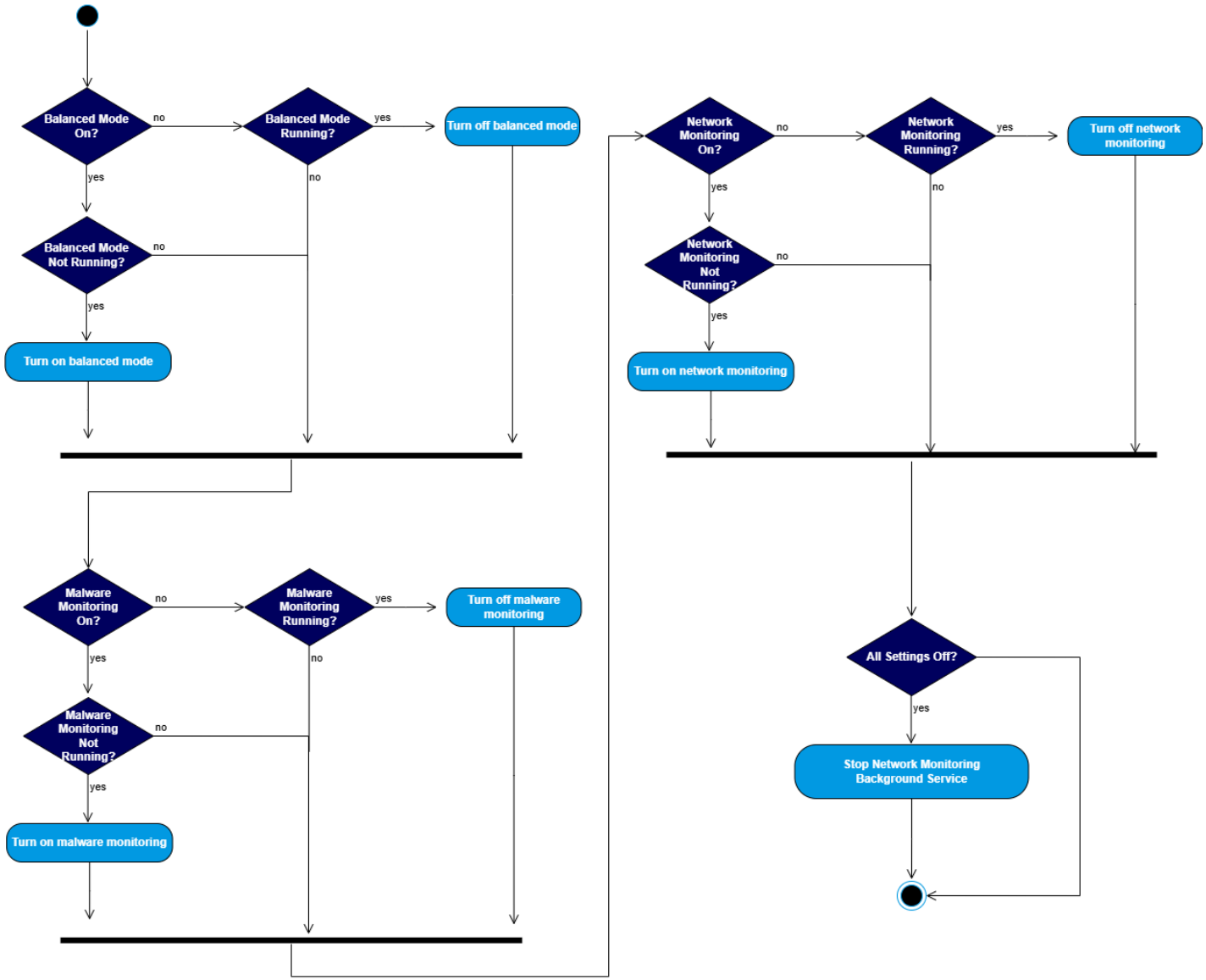

Figure 4.9 Real-time System Monitoring Activity Flow Diagram

4.4.6. Threat Remediation Module

This **Threat Remediation Activity Flow** diagram illustrates the sequential process followed when a network or malware threat is detected. The flow initiates upon the reception of a threat alert. Once a threat is received, the system proceeds to **display the threat details**, providing relevant context and information to the user or automated system. Following this, the system **offers remediation actions**, allowing for responsive measures to mitigate the detected threat. A decision point then checks whether a remediation **action has been selected**. If no action is chosen, the process loops back, awaiting user intervention. If an action is selected, the system proceeds to **perform the remediation action**, effectively addressing the threat. The workflow concludes upon completion of the remediation, ensuring a responsive and user-driven threat handling process. This flow is crucial in enabling real-time threat management and maintaining system integrity.

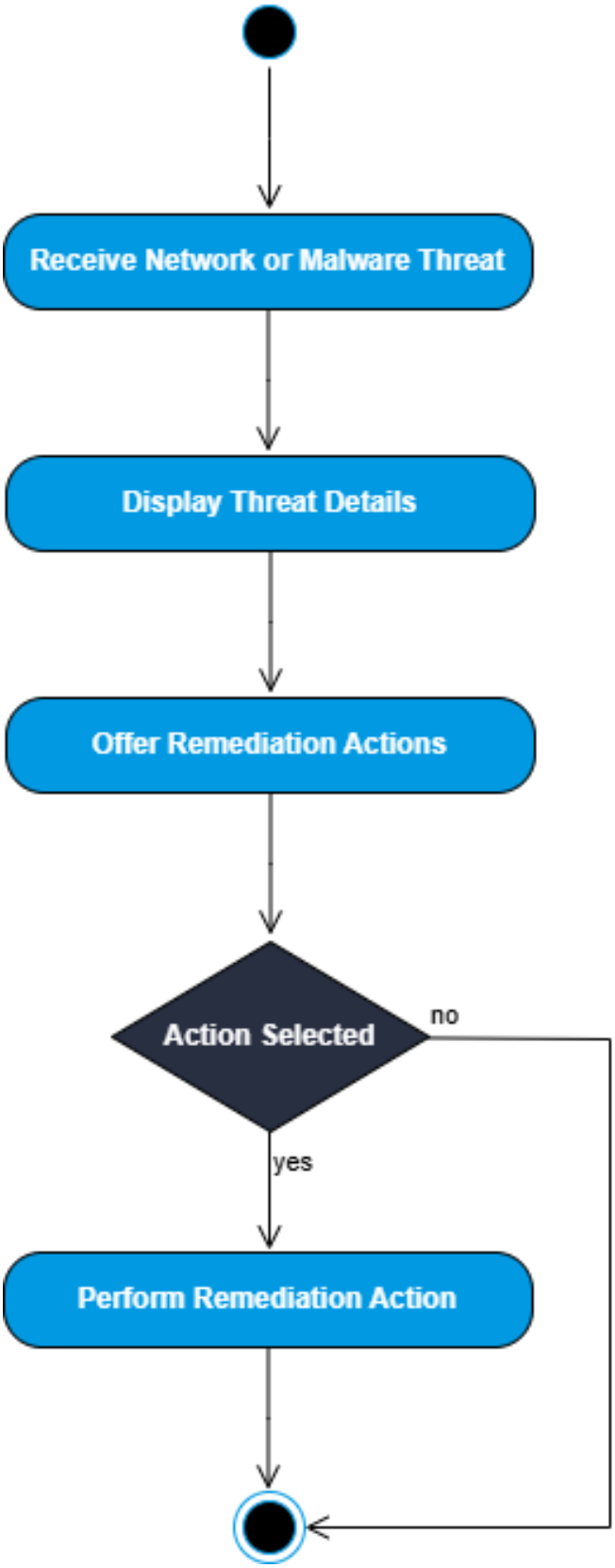

Figure 4.10 Threat Remediation Activity Flow Diagram

4.5. Use Case Diagram

This case diagram represents the functional interactions between users, system components, and the AI-Powered Anti-Malware and Application-Level Network Threat Detection System for Android devices. On the left, the **User** can perform various actions such as viewing network details, initiating manual malware scans, checking system monitoring status, selecting a security mode, viewing scan results, and interacting with a chatbot. These interactions initiate system-level use cases on the right, including **Wi-Fi scanning**, **app-level network scanning**, **malware scanning**, and **monitoring app installations**. The system also **scans flagged apps**, **retrieves threat intelligence**, and **performs security checks**. When potential threats are detected, the system **triggers the threat remediation component**, which in turn executes **malware and network threat mitigation actions** and notifies the user. Additionally, the system can **start or stop system monitoring** depending on user preferences or predefined security modes. This layered use case structure, enhanced by extensions and inclusions, ensures a responsive, user-driven approach to Android device protection while leveraging AI and network intelligence for proactive threat management.

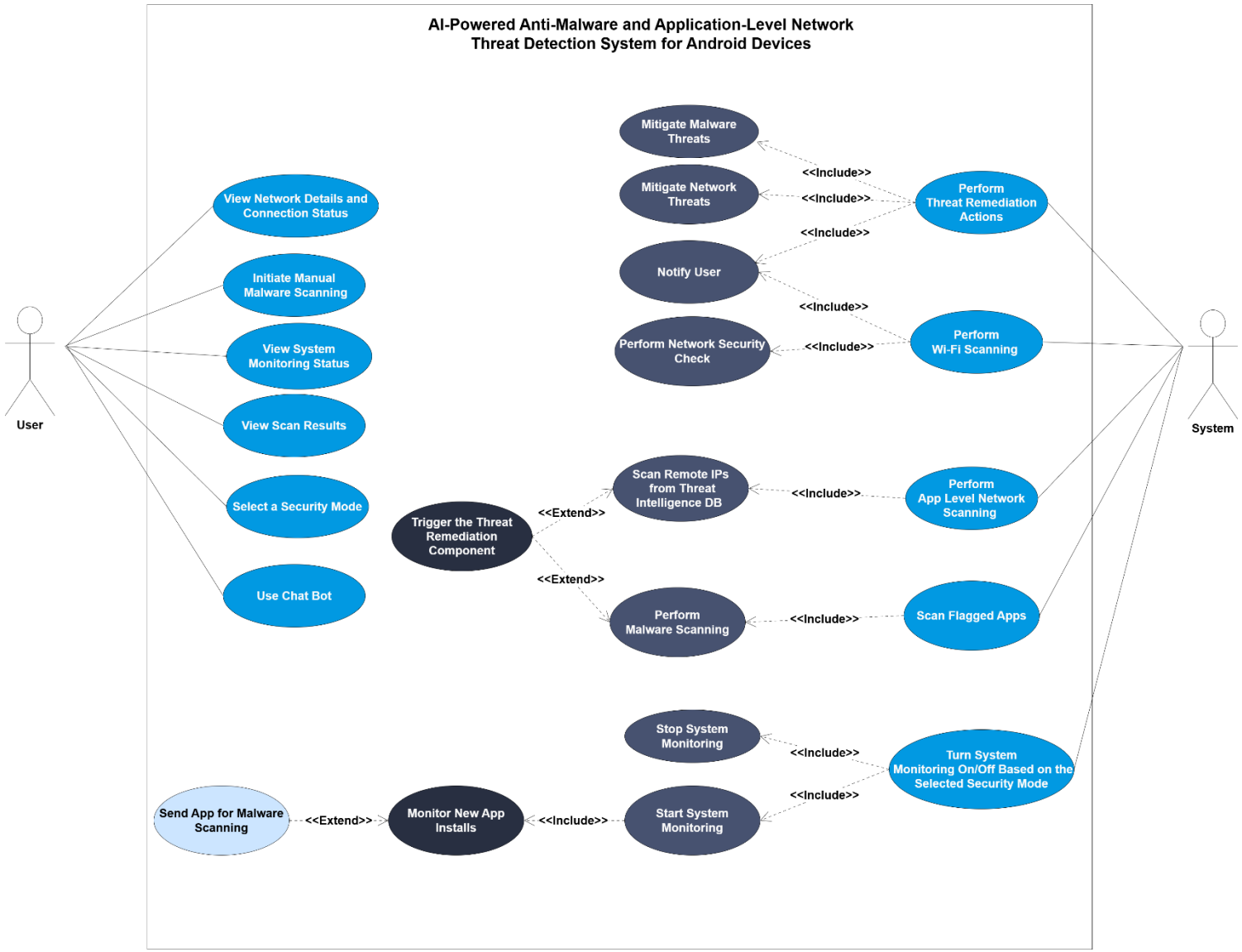

Figure 4.11 Use Case Diagram

4.6. User Interface (UI)

The user interface (UI) of the Patronus app is designed to be simple, user-friendly, and visually appealing. Each part of the interface is carefully thought out to make navigating the app smooth and intuitive.

This section explores the key components of the app's UI, highlighting the design principles and functionality of each screen. Screenshots and descriptions are included to provide a clear understanding of how the interface supports the app’s purpose and enhances the overall user experience.

4.6.1. Splash Screen

This is the splash screen, serves as the initial interface upon launching the Patronus app. This screen typically showcases the app's logo, creating an initial impression and setting the tone for the user experience.

Figure 4.12 Patronus - Splash Screen

4.6.2. Home Screen

The Home Screen of Patronus serves as the central dashboard for users, providing a quick overview of the device's current security posture. It prominently displays whether any network-based threats have been detected, offering immediate visual feedback to the user. In addition, it presents detailed information about the connected Wi-Fi network, including the SSID, BSSID, encryption type, and a security status classification derived from the app's internal scan. The screen also reflects the current state of the system monitoring feature—indicating whether it is enabled, disabled, or operating in balanced mode. For users seeking more in-depth interaction, a direct link to the Network Center screen is provided, enabling manual network scans and further analysis. A persistent bottom navigation bar is also available, granting streamlined access to the Malware Scan, Settings, and HelpBot screens, thereby enhancing usability and ensuring seamless navigation throughout the application.

Figure 4.13 Patronus Home Screen (With Threats)

4.6.3. Network Center Screen

The Network Center Screen offers users the ability to manually initiate network diagnostics through a set of three distinct scanning modes, each designed to fulfill a specific purpose. The first mode, **General Test**, provides detailed insights into the current network environment, including the SSID, BSSID, IP address, encryption type, and an overall security assessment based on predefined heuristics. The second mode, **Speed Test**, measures real-time upload and download speeds, helping users evaluate network performance. Finally, the **Troubleshoot mode** performs a basic connectivity check by pinging a well-known external server (8.8.8.8), allowing the user to quickly identify potential connection issues. This modular design ensures that users have accessible tools for both informational and diagnostic purposes, reinforcing Patronus' goal of proactive network awareness.

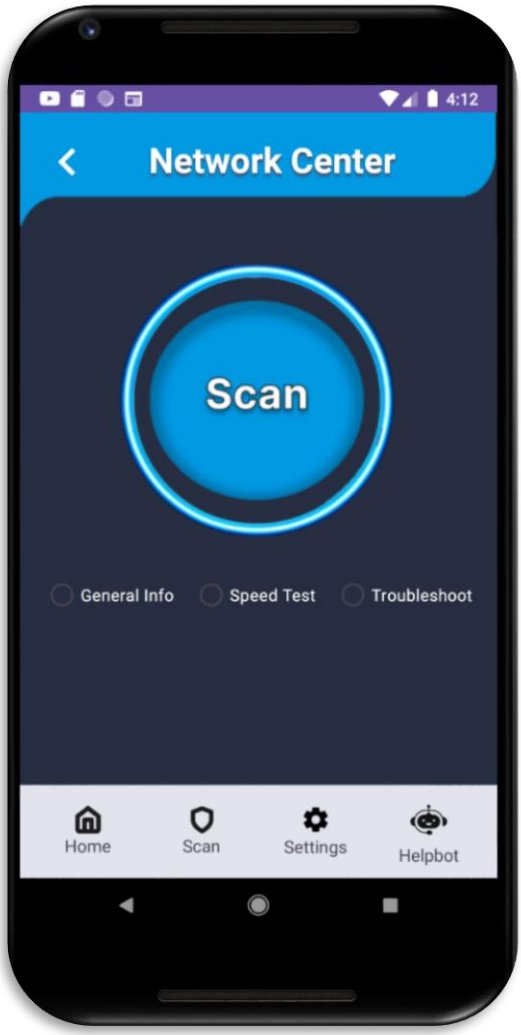

Figure 4.14 Patronus- Network Center Screen

4.6.4. Malware Scanning Screen

This screen is the device scan interface. It presents users with two scanning options: "All Apps" for a comprehensive scan of all installed applications, and "Select Apps" for a targeted scan of specific applications. The prominent "Scan" button initiates the chosen scanning process.

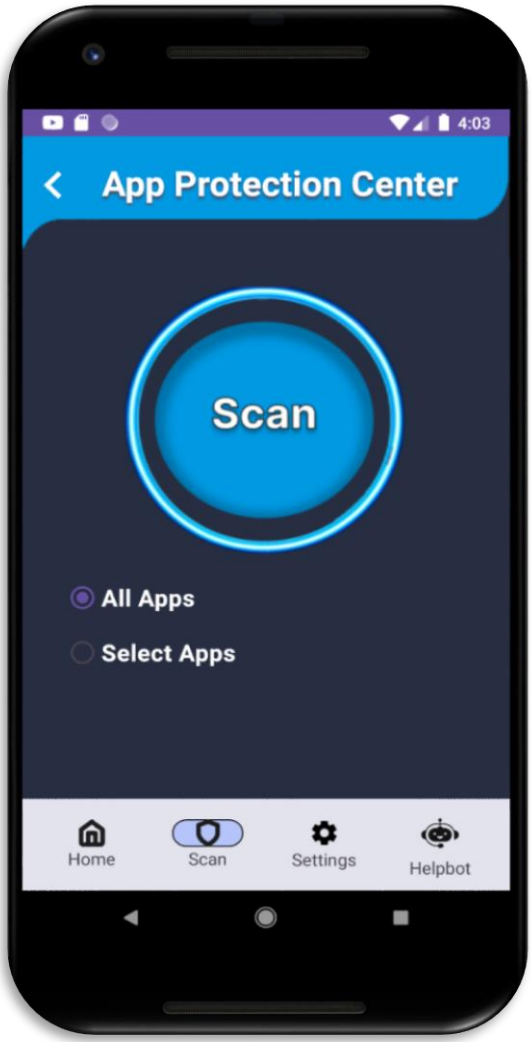

Figure 4.15 Patronus- Malware Scan Screen

4.6.5. Settings Screen

The Settings Screen in Patronus empowers users with granular control over the application's background monitoring features, promoting both personalization and a thoughtful balance between security and device performance. Users can tailor the behavior of system monitoring through intuitive toggle switches, allowing them to enable or disable specific functions according to their preferences. The available options include Background Network Monitoring and Background Malware Monitoring, each of which operates persistently in the background when enabled. Additionally, a Balanced Malware Monitoring mode is provided, offering an intelligent, adaptive approach: when activated, the app autonomously adjusts background malware scanning based on contextual device states, such as idle periods or charging cycles. This flexibility ensures that the application aligns with the user's performance expectations while maintaining robust protective measures.

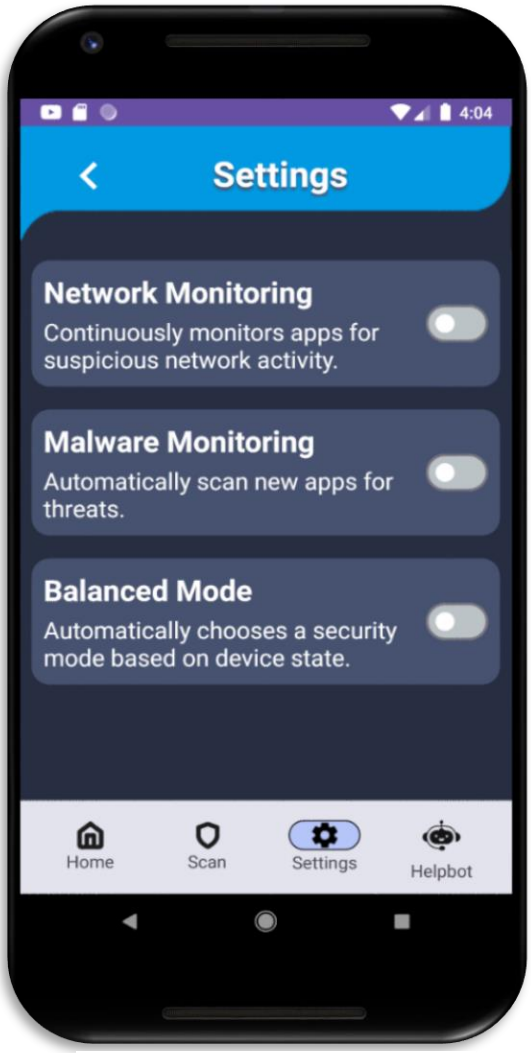

Figure 4.16 Patronus- Settings Screen

4.6.6. HelpBot Screen

The HelpBot Screen introduces an interactive support feature designed to enhance user assistance beyond traditional static documentation. Instead of relying on a conventional FAQ section, Patronus integrates a dynamic chatbot interface that allows users to ask questions in natural language and receive relevant, context-aware responses. This approach not only improves accessibility for non-technical users but also streamlines the support experience by addressing specific user concerns in real time. The HelpBot serves as a centralized knowledge assistant, capable of providing guidance on application features, security alerts, and recommended actions—ultimately contributing to a more intuitive and user-friendly environment.

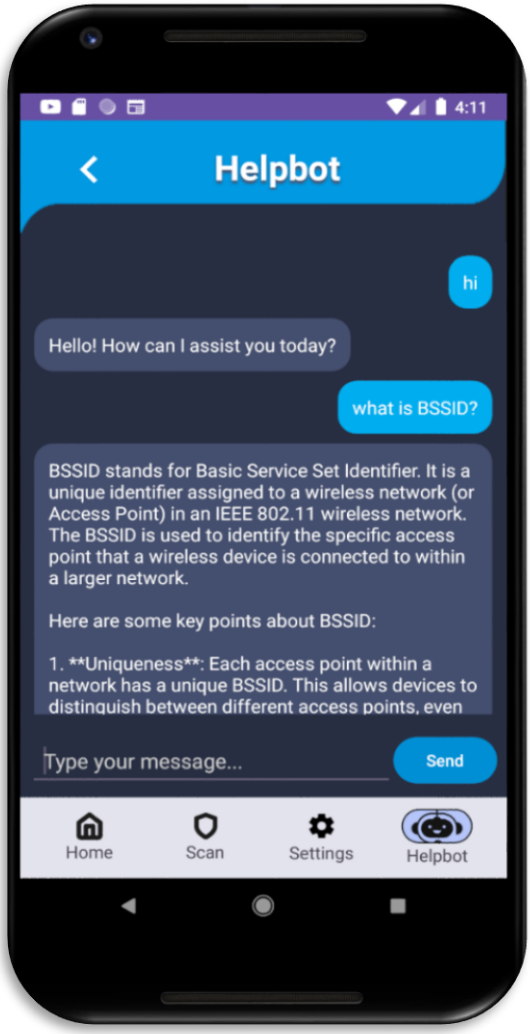

Figure 4.17 Patronus- HelpBot Screen

4.6.7. Threat Remediation Screen

The Threat Remediation feature in our app is designed to provide intelligent, context-aware responses when a potential threat is detected. When the system identifies a malicious or suspicious application, the interface displays detailed threat information along with a direct "Delete App" button, enabling users to remove the harmful app immediately from their device. However, if the detected threat originates from a web browser rather than an installed application—such as a phishing page or a malicious URL—the app adapts accordingly by omitting the delete option and instead offers a warning message. This ensures users are guided appropriately depending on the nature of the threat while maintaining a clean, user-friendly experience tailored to both app-based and browser-based risks.

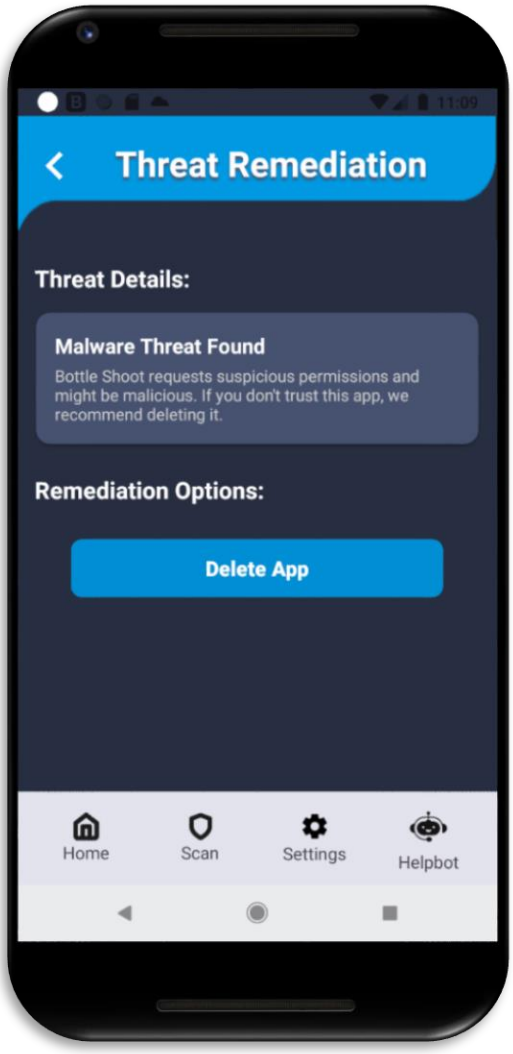

Figure 4.18 Malware Threat Remediation Screen

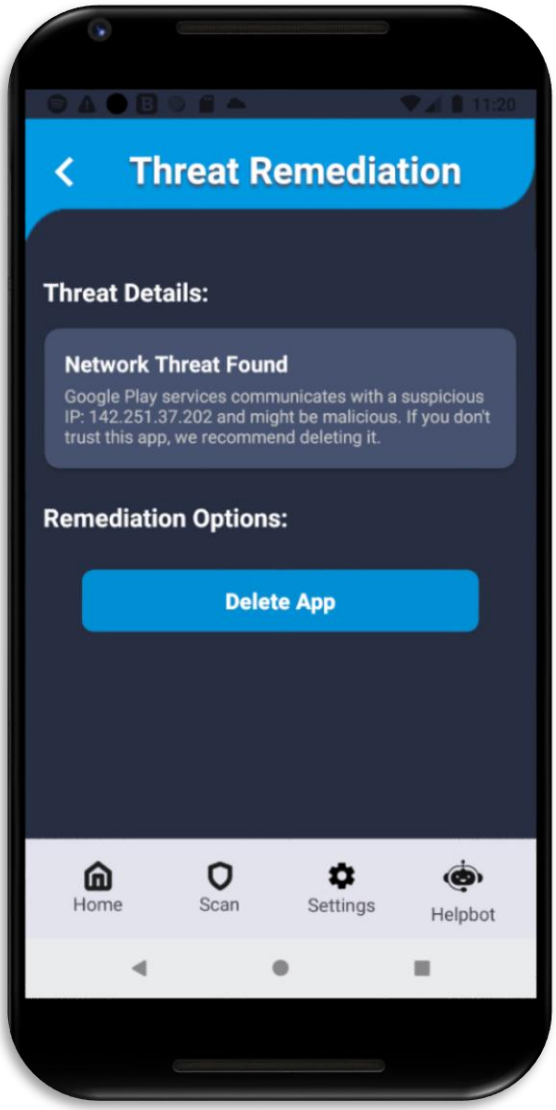

Figure 4.19 Network Threat Remediation

Note: Google Play Services does *not* actually communicate with the suspicious IP address shown in Figure 4.20. That IP was temporarily added to our threat intelligence database solely for testing purposes.

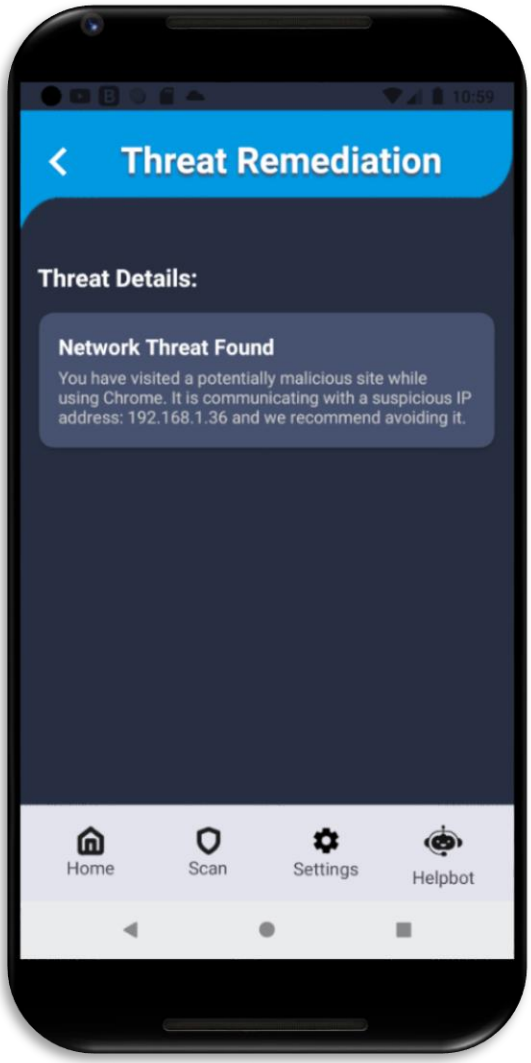

Figure 4.20 Network Threat Remediation Screen (Browsers)

4.7. Conclusion

In this chapter, we translated theoretical concepts and technical foundations into a practical application designed to improve Android device security. Each module was crafted to address specific security challenges, from proactive network analysis to real-time malware detection using AI. The modular design ensures scalability and ease of use, while the intuitive user interface enhances accessibility for non-technical users. This implementation exemplifies how modern AI and user-centered design principles can converge to offer a robust solution in the evolving domain of mobile cybersecurity.

Chapter 5

Conclusion and Future Work

5.1. Conclusion

This thesis presented the design and development of *Patronus*, an intelligent Android security application that combines machine learning–based malware detection with real-time network monitoring and threat remediation capabilities. We began by outlining Android Security Model, analyzing existing permission models, and surveying prevalent malware types. A thorough comparative analysis of commercial mobile security solutions highlighted key gaps in existing tools—particularly the lack of transparency, on-device intelligence, and user-centric design. To address these issues, we trained a neural network using multiple datasets (DREBIN, TUANDROMD, COLCOM) and engineered permission-based features to achieve high generalization performance against unseen malware. We further integrated Patronus into a modular Android app, incorporating components such as the Network Center, HelpBot assistant, and dynamic threat remediation workflows. Extensive evaluations validated the system's capability to detect threats with precision while remaining lightweight and user-friendly. Ultimately, Patronus demonstrates that integrating AI-driven detection with contextual awareness and system-level visibility can provide more robust and proactive mobile security. This project not only contributes a novel solution but also establishes a flexible foundation for future research in adaptive, on-device threat defense.

5.2. Insights from Phase 2

Phase 2 focused on implementing the system that was conceptualized and documented during Phase 1. While the initial phase laid the foundation through extensive research, design planning, and architectural documentation, Phase 2 brought those ideas into practice. We developed the *Patronus* Android application, integrating the AI-powered malware detection model, real-time network monitoring, and user-facing security features such as HelpBot and the threat remediation module. The UI components and module workflows were implemented to reflect the designs and logic described previously. This phase also included training and evaluating the neural network model using datasets like DREBIN and TUANDROMD, enabling us to assess its performance in real deployment scenarios. By translating design into code, Phase 2 not only validated the feasibility of our approach but also surfaced valuable insights around system integration, user experience, and practical limitations—guiding future enhancements of the application.

5.3. Future Work

Looking ahead, we plan to extend the application-level network scanner support to devices running Android 10 and above by exploring alternative approaches that comply with newer security restrictions, such as using VPN-based traffic inspection or Android's newer network monitoring APIs. Additionally, we aim to enhance the HelpBot module by integrating a voice interaction feature, enabling users to receive security guidance and perform key actions through voice commands. This will make Patronus more accessible, user-friendly, and responsive—especially for users who may prefer hands-free interaction or require assistive features.

Appendix A - Code Snippets

This appendix includes selected source code snippets relevant to the implementation of Patronus. The full codebase is available at https://github.com/nouranabdlsalam/patronus .

A.1. Wi-Fi Scanner

This function calculates a Wi-Fi risk score based on encryption type, signal strength, and SSID confusion. It assigns higher scores to weaker or open networks (like WEP or ESS) and suspiciously strong signals. It also flags duplicate SSIDs with different security types, capping the total risk at 10.

```
public int calculateRiskScore(ScanResult scanResult, List<ScanResult> allResults) {
    int score = 0;
    String capabilities = scanResult.capabilities;
    int rssi = scanResult.level;
    String ssid = scanResult.SSID;
    if (capabilities.contains("WEP")) {
        score += 4; // High risk
    } else if (capabilities.contains("WPA") && !capabilities.contains("WPA2")) {
        score += 3; // Moderate risk
    } else if (capabilities.contains("WPA2") && !capabilities.contains("WPA3")) {
        score += 2; // Lower risk
    } else if (capabilities.contains("WPA3")) {
        score += 1; // Low risk
    } else if (capabilities.contains("ESS")) {
        score += 5; // Open network, very high risk
    }
    if (rssi > -40 && capabilities.contains("ESS")) {
        score += 2; // suspiciously strong open AP
    }
    for (ScanResult other : allResults) {
        if (!other.BSSID.equals(scanResult.BSSID) && other.SSID.equals(ssid)) {
            if (!getSecurityType(other.capabilities).equals(getSecurityType(capabilities))) {
                score += 2; // suspicious SSID confusion
            }
        }
    }
    return Math.min(score, 10);
}
```

Figure 6.1 Wi-Fi Risk Scoring Method

This code evaluates internet connection quality based on upload and download speeds. It classifies the connection as Excellent, Moderate, or Poor, and provides a matching suggestion. The result is displayed in a dialog with a formatted message.

```
speedChecker.measureUploadSpeed( testUrl: "https://httpbin.org/post", uploadSpeed -> runOnUiThread(() -> {
    uploadSpeedValue[0] = uploadSpeed;
    speedResults.append("🚀 Upload Speed: ")
            .append(String.format("%.2f", uploadSpeed)).append(" Mbps\n");

    String quality;
    String suggestion;
    if (downloadSpeedValue[0] >= 25 && uploadSpeedValue[0] >= 5) {
        quality = "✅ Excellent Connection";
        suggestion = "Your internet is great! No action needed.";
    } else if (downloadSpeedValue[0] >= 5 && uploadSpeedValue[0] >= 1) {
        quality = "⚠️ Moderate Connection";
        suggestion = "Try moving closer to the router or reducing background usage.";
    } else {
        quality = "❌ Poor Connection";
        suggestion = "Consider restarting your router, or upgrading your plan.";
    }

    speedResults.append("\n📊 Connection Quality: ").append(quality).append("\n")
            .append("💡 Suggestion: ").append(suggestion);
    dialog.setMessage(speedResults.toString());
}));
```

Figure 6.2 Speed Checker Code Snippet

A.2. Application-Level Network Scanner

This method scanConnections() scans user-installed apps to identify those communicating with known malicious IP addresses. It checks each app's IPs against a database and returns a list of apps found to connect to malicious hosts.

```
public ArrayList<App> scanConnections(ArrayList<App> userApps){
    ArrayList<App> maliciousApps = new ArrayList<>();
    IPDatabase db = IPDatabase.getInstance(context);
    IPDao ipDao = db.ipDao();
    for (App app : userApps) {
        if (!app.getIPs().isEmpty()){
            for (String ip: app.getIPs()) {
                boolean isMalicious = ipDao.isMalicious(ip);
                if (isMalicious){
                    app.maliciousIPs.add(ip);
                    if (!maliciousApps.contains(app)){
                        maliciousApps.add(app);
                    }
                }
            }
        }
    }
    return maliciousApps;
}
```

Figure 6.3 Application-Level Connection Scanner Method

A.3. AI-powered Malware Detection Module

This code snippet implements a context-aware scanning function for Android applications. The scanApp method analyzes an app by extracting its features, classifying it, determining its category, and calculating its sideloading score. These parameters are passed to the getContextAwareScore function, which computes a weighted risk score based on the model confidence, category risk, and sideloading score. The score is then evaluated to determine the app's potential risk level. Debug logs are used to trace the calculation steps and final score.

```
protected int scanApp(App app) throws PackageManager.NameNotFoundException {
    app.setFeatures(extractFeatures(app));
    double modelConfidence = classify(app);
    int category = getAppCategory(app);
    int sideloadingScore = getSideloadingScore(app);
    return getContextAwareScore(modelConfidence, category, sideloadingScore);
}

1 usage  new *
public int getContextAwareScore(double modelConfidence, int category, int sideloadingScore){
    double categoryRisk = getCategoryRisk(category);
    double contextAwareScore = 0.5 * modelConfidence + 0.3 * categoryRisk + 0.2 * sideloadingScore;
    Log.d( tag: "CONTEXT",  msg: "model confidence: " + modelConfidence + ", category: " + category +
            ", sideloading score: " + sideloadingScore + ", context score: " + contextAwareScore);
    return contextAwareScore > 0.7 ? 1 : 0;
}
```

Figure 6.4 Malware Scanning Function and Context-Aware Scoring Function

A.4. Security Modes and Network Monitoring Settings

This code snippet manages the behavior of toggle switches for system monitoring modes. It includes three switches: Malware Monitoring Switch: Enables malware monitoring and automatically disables the balanced mode if selected. Network Monitoring Switch: Enables network monitoring independently. Balanced Mode Switch: Activates a balanced monitoring mode and automatically disables malware monitoring if selected. Each switch updates the app’s preferences and restarts the system monitoring service to apply the changes in real-time.

```
malwareSwitch.setOnCheckedChangeListener((buttonView, isChecked) -> {
    prefs.setMalwareMonitoringOn(isChecked);
    if (isChecked){
        balancedSwitch.setChecked(false);
        prefs.setBalancedModeOn(false);
    }
    restartSystemMonitoringService( context: this);
});


networkSwitch.setOnCheckedChangeListener((buttonView, isChecked) -> {
    prefs.setNetworkMonitoringOn(isChecked);
    restartSystemMonitoringService( context: this);
});

balancedSwitch.setOnCheckedChangeListener((buttonView, isChecked) -> {
    prefs.setBalancedModeOn(isChecked);
    if (isChecked){
        malwareSwitch.setChecked(false);
        prefs.setMalwareMonitoringOn(false);
    }
    restartSystemMonitoringService( context: this);
});
```

Figure 6.5 Security Monitoring and Network Monitoring Settings

This code snippet demonstrates the activation of three key monitoring features in an Android application: Malware Monitoring: Registers a broadcast receiver for package installation and replacement events to monitor potential malware-related activities. Network Monitoring: Starts a periodic network monitoring task using a handler. Auto Mode Activation: Initiates a balanced operational mode through periodic task execution. Each function logs its activation status for debugging and confirmation purposes.

```
public void turnMalwareMonitoringOn(){
    if (!isPackageReceiverRegistered){
        IntentFilter filter = new IntentFilter();
        filter.addAction(Intent.ACTION_PACKAGE_ADDED);
        filter.addAction(Intent.ACTION_PACKAGE_REPLACED);
        filter.addDataScheme("package");
        registerReceiver(packageInstallReceiver, filter);
        isPackageReceiverRegistered = true;
        Log.d( tag: "SETTINGS",  msg: "Malware Monitoring is on");
    }
}
1 usage   nouranabdlsalam
public void turnNetworkMonitoringOn(){
    handler.post(periodicNetworkTask);
    Log.d( tag: "SETTINGS",  msg: "Network Monitoring is on");
}
1 usage   nouranabdlsalam
public void turnAutoModeOn() { handler.post(periodicBalancedMode); }
```

Figure 6.6 System Monitoring Service (Turning on Settings)

This code disables three monitoring features: Malware Monitoring: unregisters a receiver for app install events. Network Monitoring: stops a periodic network task. Auto Mode: deactivates balanced operation tasks. Each action is logged for debugging and status tracking.

```
public void turnMalwareMonitoringOff(){
    if (isPackageReceiverRegistered){
        if (packageInstallReceiver != null) {
            unregisterReceiver(packageInstallReceiver);
            isPackageReceiverRegistered = false;
            Toast.makeText( context: this,  text: "Unregistered Receiver", Toast.LENGTH_SHORT).show();
            Log.d( tag: "SETTINGS",  msg: "Malware Monitoring is off");
        }
    }
}
2 usages   nouranabdlsalam
public void turnNetworkMonitoringOff(){
    handler.removeCallbacks(periodicNetworkTask);
    Log.d( tag: "SETTINGS",  msg: "Network Monitoring is off");
}
2 usages   nouranabdlsalam
public void turnAutoModeOff() { handler.removeCallbacks(periodicBalancedMode); }
```

Figure 6.7 System Monitoring Service (Turning off Settings)

This code provides utility functions for monitoring battery conditions: getBatteryLevel(): Returns the current battery percentage. isPowerSaveMode(): Checks if the device is in power-saving mode. isDeviceCharging(): Determines if the device is currently charging or fully charged. These methods help adapt app behavior based on battery state.

```
private int getBatteryLevel() {
    BatteryManager bm = (BatteryManager) getSystemService(BATTERY_SERVICE);
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
        return bm.getIntProperty(BatteryManager.BATTERY_PROPERTY_CAPACITY);
    }
    return -1; // Unsupported on very old devices
}

1 usage  nouranabdlsalam
private boolean isPowerSaveMode() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
        PowerManager pm = (PowerManager) getSystemService(POWER_SERVICE);
        return pm.isPowerSaveMode();
    }
    return false;
}

2 usages  nouranabdlsalam
private boolean isDeviceCharging() {
    IntentFilter ifilter = new IntentFilter(Intent.ACTION_BATTERY_CHANGED);
    Intent batteryStatus = registerReceiver( receiver: null, ifilter);
    int status = batteryStatus != null ? batteryStatus.getIntExtra(BatteryManager.EXTRA_STATUS,  defaultValue: -1) : -1;
    return status == BatteryManager.BATTERY_STATUS_CHARGING || status == BatteryManager.BATTERY_STATUS_FULL;
}
```

Figure 6.8 Security Modes: Battery Monitoring

A.5. Threat Remediation

This code enables the "Delete App" button to trigger the uninstallation of a detected malicious app. It creates an intent with the ACTION_UNINSTALL_PACKAGE action, targets the app's package name, and launches the system uninstall prompt when clicked.

```
    option1.setText("Delete App");
    nouranabdlsalam
    option1.setOnClickListener(new View.OnClickListener() {
        nouranabdlsalam
        @Override
        public void onClick(View v) {
            Intent intent = new Intent(Intent.ACTION_UNINSTALL_PACKAGE);
            intent.setData(Uri.parse( uriString: "package:" + MaliciousApp.getPackageName()));
            intent.putExtra(Intent.EXTRA_RETURN_RESULT,  value: true);
            startActivity(intent);
        }
    });

}
```

Figure 6.9 Threat Remediation: Deleting Application

A.6. HelpBot

This code snippet demonstrates how to send a POST request to an API endpoint using HttpURLConnection. It sets the required headers for authorization and JSON content type, and builds a JSON request body that includes the model name, user input messages, temperature, and maximum token limit. The request is then written to the API using a BufferedWriter with UTF-8 encoding. The API request payload is also logged for debugging purposes.

```
try {
    URL url = new URL(API_URL);
    HttpURLConnection connection = (HttpURLConnection) url.openConnection();
    connection.setRequestMethod("POST");
    connection.setRequestProperty("Authorization", API_TOKEN);
    connection.setRequestProperty("Content-Type", "application/json");
    connection.setDoOutput(true);
    JSONObject jsonInput = new JSONObject();
    jsonInput.put( name: "model",  value: "mistral/ministral-8b"); // must match OpenRouter model name
    JSONArray messages = new JSONArray();
    JSONObject userMessage = new JSONObject();
    userMessage.put( name: "role",  value: "user");
    userMessage.put( name: "content", params[0]);
    messages.put(userMessage);
    jsonInput.put( name: "messages", messages);
    jsonInput.put( name: "temperature",  value: 0.7); // optional
    jsonInput.put( name: "max_tokens",  value: 300); // optional
    JSONObject userMsg = new JSONObject();
    userMsg.put( name: "role",  value: "user");
    userMsg.put( name: "content", params[0]);
    messages.put(userMsg);
    jsonInput.put( name: "messages", messages);
    Log.d( tag: "API Request JSON", jsonInput.toString());
    OutputStream os = connection.getOutputStream();
    BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(os,
            charsetName: "UTF-8"));
    writer.write(jsonInput.toString());
    writer.flush();
    writer.close();
    os.close();
```

Figure 6.10 HelpBot API Request

References

[1] S. Sharma, R. Kumar, and C. RamaKrishna, "A survey on analysis and detection of Android ransomware," Concurrency and Computation: Practice and Experience, vol. 34, no. 9, pp. e6272, 2022. DOI: 10.1002/cpe.6272

[2] H. A. Al-Ofeishat, "Enhancing Android security: Network-driven machine learning approach for malware detection," Journal of Theoretical and Applied Information

[3] M. Almahmoud, D. Alzu'bi, and Q. Yaseen, "ReDroidDet: Android Malware Detection Based on Recurrent Neural Network," Proc. 2nd Int. Workshop Data-Driven Security (DDSW 2021), Warsaw, Poland, 2021, pp. 1–8.

[4] W. Wang, M. Zhao, and J. Wang, "Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network," Journal of Ambient Intelligence and Humanized Computing, vol. 10, no. 7, pp. 3035-3043, 2019.

[5] S. Y. Yerima and S. Sezer, “Machine learning for Android malware detection using permission and API calls,” in Proc. 2017 Int. Conf. Inf. Commun. Syst. (ICICS), 2017, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/8245867

[6] A. Voyiatzis, C. Fidas, L. Mitrou, and A. Papanikolaou, “The 3rd International Workshop on Information Systems Security Engineering (WISSE 2013),” ICSdWeb, May 2013. [Online]. Available: https://icsdweb.aegean.gr/awid/awid3.

[7] F. Diro, N. Chilamkurti, and I. Ahmed, "A deep learning-based security framework for fog-enabled industrial Internet of Things," IEEE Transactions on Industrial Informatics, vol. 18, no. 8, pp. 5445-5452, Aug. 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9797689.

[8] AV-TEST. "16 Android Security Apps vs. Google Play Protect in an Endurance Test." AV-TEST, 2023. [Online]. Available: https://www.av-test.org/en/news/16-android-security-apps-vs-google-play-protect-in-an-endurance-test/.

[9] P. Borah, D. K. Bhattacharyya, and J. K. Kalita, “Malware dataset generation and evaluation,” in Proc. 2020 IEEE 4th Conf. Inf. Commun. Technol. (CICT), 2020, pp. 1–6.

[10] S. Yerima, Android malware dataset for machine learning 2, figshare, 2018. [Online]. Available: https://doi.org/10.6084/m9.figshare.5854653.v1