University of Passau
Faculty of Computer Science and Mathematics

Chair of Computer Engineering
Prof. Dr. Stefan Katzenbeisser

Master's Thesis
in
Computer Science

Android Threat Detection Through Passive VPN Monitoring and IP Reputation Analysis

Shayan Rostamzadeh
111769

Date:	2025-11-05
Supervisors:	Prof. Dr. Stefan Katzenbeisser
	Dr. Ing. Nikolaos Athanasios Anagnostopoulos
Advisor:	Nico Mexis

Rostamzadeh, Shayan
Theresienstrasse 8
94032, Passau

ERKLÄRUNG

Ich erkläre, dass ich die vorliegende Arbeit mit dem Titel „Android Threat Detection Through Passive VPN Monitoring and IP Reputation Analysis“ selbstständig, ohne unzulässige Hilfe und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst habe und dass alle wörtlich oder sinngemäß übernommenen Stellen als solche kenntlich gemacht sind.

Mit der aktuell geltenden Fassung der Satzung der Universität Passau zur Sicherung guter wissenschaftlicher Praxis und für den Umgang mit wissenschaftlichem Fehlverhalten vom 31. Juli 2008 (vABlUP Seite 283) bin ich vertraut.

Ich erkläre mich mit einer Überprüfung der Arbeit unter Zuhilfenahme von Dienstleistungen Dritter (z.B. Anti-Plagiatssoftware) zur Gewährleistung der einwandfreien Kennzeichnung übernommener Ausführungen ohne Verletzung geistigen Eigentums an einem von anderen geschaffenen urheberrechtlich geschützten Werk oder von anderen stammenden wesentlichen wissenschaftlichen Erkenntnissen, Hypothesen, Lehren oder Forschungsansätzen einverstanden.

.....
(Name, Vorname)

Translation of German text (notice: Only the German text is legally binding)

I hereby confirm that I have composed the present scientific work entitled “Android Threat Detection Through Passive VPN Monitoring and IP Reputation Analysis” independently without anybody else’s assistance and utilising no sources or resources other than those specified. I certify that any content adopted literally or in substance has been properly identified and attributed.

I have familiarised myself with the University of Passau’s most recent Guidelines for Good Scientific Practice and Scientific Misconduct Ramifications of 31 July 2008 (vABlUP page 283).

I declare my consent to the use of third-party services (e.g. anti-plagiarism software) for the examination of my work to verify the absence of impermissible representation of adopted content without adequate attribution, which would violate the intellectual property rights of others by claiming ownership of somebody else’s work, scientific findings, hypotheses, teachings or research approaches.

Supervisor contacts:

Prof. Dr. Stefan Katzenbeisser
Chair of Computer Engineering
University of Passau
Email: stefan.katzenbeisser@uni-passau.de
Web: https://www.fim.uni-passau.de/en/computer-engineering/

Dr. Ing. Nikolaos Athanasios Anagnostopoulos
The chair of your second advisor professor
University of Passau
Email: nikolaos.anagnostopoulos@uni-passau.de
Web: https://www.anagnostopoulos.academy/

Abstract

Mobile devices are becoming more and more immersed in our daily lives, making them attractive targets for threats such as malware, data exfiltration, and unauthorized access and even end-points for APTs (Advanced Persistent Threat) in a huge number of companies that comply with with BYOD (Bring Your Own Device) policy for cost reduction and personal convenience. The comprehensive use of mobile devices in sensitive and vital business operations highlights the necessity of advanced monitoring and threat detection mechanisms.

While existing tools like PCAPdroid and Ant-Monitor provide traffic analysis and monitoring capabilities, they often lack integration with real-time threat recognition. This project exhibits the design and implementation of an Android-based threat detection application that leverages the Android VPNService API to capture and intercept network/internet traffic. This comes alongside the functionality to map associated packets to originating device applications. This thesis project incorporates AbuseIPDB. A well-known platform dedicated to helping users and administrators combat the spread of hackers, spammers, and abusive activity on the internet. This incorporation is to assess the maliciousness of destination IP addresses in the outgoing internet packets, notifying the user of the corresponding potential risk(s) that the application can introduce.

This application is developed as a complementary extension to PCAPdroid that lacks live threat detection and analysis of network/internet traffic. It utilizes the passive packet-capture capabilities of PCAPdroid and employs AbuseIPDB capabilities to bridge the gap between packet capture and live threat analysis combined with the latest modern user interface approaches.

This application receives the outgoing IP address, application UIDs to extract the app-specific information alongside other useful data in the form of a PCAPNG file via a local TCP Server from PCAPdroid and subsequently transmits and inquiry to AbuseIPDB to evaluate the maliciousness of outbound traffic.

Threat Detector illustrates an ability to identify suspicious connections with minimal performance and storage overhead, highlighting it as a potent and practical tool to enhance mobile security, privacy and user awareness.

Contents

1 Introduction

1.1 Background and Motivation

In today's connected world, mobile devices have evolved from simple communication intermediaries to vital hubs not only for personal use but also professional activities. They have undoubtedly become personal assistants, banking platforms, health trackers and entertainment centers, and also professional workstations. Smartphones, Tablets, and other similar portable devices now store sensitive information such as personal messages, financial intel, business-related documents and login credentials. Nowadays as mobile devices are increasingly integrating with enterprise businesses and consequently their associate networks through policies such as BYOD (Bring Your Own Device), we see them more frequently being subjected to cyber attacks including mobile malware, unauthorized access, data exfiltration, Advanced Persistent Threats (APTs), etc. This widespread adaption of mobile technology and its undeniable integration in our daily personal and professional lives in combination with users and companies reliance have considerably expanded the attack surface for adversaries. Additionally, the on-going increase in the use of mobile devices to access sensitive corporate and financial resources also amplifies the potential damage an intrusion can lead to. This means that the security landscape of mobile platforms is therefore, highly critical. Thus, it requires solutions and approaches that continuously adapt to such evolving threats while ensuring practicality.

1.2 Problem Statement

Most of enterprise network systems belong to a pool of PCs (Personal computers) and servers. Therefore, the majority of traditional cybersecurity measures often focus on such systems. However, the integration and usage of mobile devices in enterprise networks introduce unique challenges that require a different approach. The diversity of mobile devices' operating systems, varying security and privacy policies, constant updates and patches, and openness of certain app-ecosystems complicate protecting mobile devices. Among mobile operating systems, Android has gained the most popularity and market share due to its open-source nature and flexibility to be implemented in various environments. However, Android's open-source architecture, alongside its fragmented ecosystem and, in a lot of cases, its inconsistent update policies make it in particular considerably susceptible to attacks. These lead malicious actors to utilize various application-level and network-based attacks and also abuse hardware and software vulnerabilities to compromise user's privacy and the organization's security.

It is worth mentioning that the traditional endpoint security solutions such as anti-malware software, often lead to inadequate results for mobile devices including Android phones. Many are based on signature recognition, and operate reactively, meaning they typically identify malware signatures after the infection completely took place. Moreover, they are incapable of monitoring the full scope of the device's network and its behaviour. Furthermore, Android system suffers from an invisibility gap. Even though Android has a sandboxing mechanism which provides isolation between running applications, it also limits the visibility of network activities associated with each app. This simply means that users and more importantly administrators cannot easily determine which of the running applications is connecting to external servers, nor can they evaluate the legitimacy of the established connections.

1.2.1 Data Exfiltration Risk

As the usage of mobile applications increases in business constellations and the centralization of information is more intensified, more internet connections and data transfer take place which broaden the attack surfaces on mobile devices. This would potentially open some doors for the adversaries to abuse these connections for their own benefit while user privacy is completely neglected. This mainly is because of the gaps that current security solution approaches cannot cover. A huge threat that connection of mobile applications with internet brings along, is data exfiltration. Data exfiltration is an underlying concept for most of the applications to function correctly since their logic relies on connection to a back-end server via internet. This however, can theoretically compromise user privacy if user's consent is not taken into consideration. This could take place by utilizing internet packets' outbound connections. The mobile threat detection application developed in this thesis addresses this very data transfer specifically. These challenges are addressed by combining real-time internet traffic monitoring, UID-to-application mapping, and finally threat intelligence integration.

As a conclusion, there is a significant need for a real-time, lightweight monitoring solution that not only captures the traffic, but also identifies suspicious patterns, stacked with the capability to map each of those activities to specific applications. This gives the users and administrators the visibility, without which, organizations might remain at risk of covert data leakage and eventually exposure to malicious infrastructure.

1.3 Existing Solutions and Their Limits

This paper is not the first to introduce defensive and preventive solutions to offensive security attempts made by malicious actors. However, it acts as a complementary extension for previously designed solutions such as *PCAPdroid*[1] and *Ant-Monitor*[2] that function on unrooted devices based on Android's VpnService mechanism. Such tools represent vital insights into how applications interact with local and external servers, letting the user perform forensic analysis of network traffic and point out potential anomaly-indicating behaviours.

[1] `https://github.com/emanuele-f/PCAPdroid`
[2] `https://github.com/UCI-Networking-Group/AntMonitor`

While current solutions for mobile threat detection and network monitoring are well-established and offer the foundation for capabilities such as network traffic analysis, application activity monitoring, and anomaly detection, they often:

- **Lack real-time threat detection and automated integration of threat intelligence:** Neither of the mentioned monitoring tools implement automated checks against malicious IP databases or incorporate a reputation assessment service. As a result, threat identification relies heavily on the expertise the user possesses and the manual process of analysing each and every IP address.
- **Have limited real-time protection:** While they are really effective in packet capture, they do not provide the user with any sort of alerts regarding suspicious activities.
- **Dump the device traffic as a PCAP file and send it remotely for further analysis (e.g. to Wireshark):** This satisfies the inevitable need for an external inspection system/application to allocate the IP address and the network connections to a white or black list.
- **Have limited user accessibility:** These tools as shown later in this paper, often present raw data packets. This can be significantly overwhelming for users without any technical background. The lack of intuitive, well-designed interfaces, and actionable insights introduce restrictions to adapting such applications with daily lives and limit their use to only research contexts.
- **Lack blocking capabilities:** The mentioned tools among other existing ones do not typically support active traffic blocking capabilities, leaving the user without any mitigating countermeasures once the threat is detected.

The complexity of mobile threats that are on continues growth and the aforementioned gaps highlight the vital necessity of a solution that not only makes uses of monitoring capabilities provided by mobile platforms (e.g Android's VpnService), but also delivers actionable, intelligence-driven, and user-friendly insights.

1.4 Research Objectives

This thesis aims to design and evaluate a threat detection application for Android devices that addresses some of the above-mentioned limitations.

This project aims specifically to bridge some of the gaps using the following implementations:

- **Real-time traffic monitoring** using Android's VpnService API provided by a companion application (PCAPdroid).
- **Mapping of UIDs to applications** that allows network traffic flow to be associated with a corresponding application which utilizes local/external network communication channels to assist user with improved transparency to identify which applications are communicating with malicious external sources.
- **Integration of the intelligence and reputation-checking databases such as AbuseIPDB**, enabling the application to automatically send and enquiry against the destination IP addresses.
- **User alerts** to notify users and administrators of potentially suspicious network behaviour.

- **Lightweight and efficient design and usability** that ensures the application runs smoothly and efficiently on the consumer's device without leaving any drastic performance impact.

By reaching these objectives, this project seeks to shorten the gap between raw packet monitoring and fully-integrated mobile security solutions.

From the network monitoring applications mentioned previously, PCAPdroid has been chosen as the underlying solution that provides not only the capabilities to passively sniff app-specific network traffic but also presents application metadata. According to its *official website*[3] PCAPdroid is an open source network capture and monitoring tool for Android devices which works without root privileges.

The common use cases of PCAPdroid include:

- Analyze the connections made by the apps installed on the device; both user and system apps.
- Dump the device traffic as a PCAP and send it remotely for futher analysis (e.g. to Wireshark).
- Decrypt the HTTPS/TLS traffic of a specific app. PCAPdroid leverages the Android VpnService to receive all the traffic generated by the Android apps. No external VPN is actually created, the traffic is processed locally by the app.

These objectives will be achieved by leveraging the Android's VpnService API to inspect the application-specific packets, and to correlate them with app-generated traffic and as the last step, to cross-reference the suspicious IP (Internet Protocol) addresses with external thread detection databases such as AbuseIPDB. Using this approach the visibility of potential malicious activities is enhanced and also some actionable insights are provided that eventually can assist users and organizations to mitigate risks and potential vulnerabilities before they lead to various security incidents.

This application alongside the real-time threat detection provided by the project presented in this paper will expand the possibilities and ease the user interaction and while providing notifications in case of malicious activities.

1.5 Limitations of this Project

This project is an extension of previously well-established monitoring solutions such as PCAPdroid. This means that this application will only be functional if installed and run alongside PCAPdroid as an underlying foundation for this project. Installation and utilization of PCAPdroid is essential since it simplifies the passive packet capture via Android's VpnService API and implements the VPN behaviour without disturbing the internet communication channels. Furthermore, it uses kernel-level APIs written in C++ that removes the need to root the device to be able to access the resources that are not defined in the normal user context such as accessing the virtual folder in the Android subsystem directory "/proc/net/tcp" which is vital to retrieve UIDs of application packages to show which app is using a suspicious connection. This not only guarantees the efficiency of the application but also removes the need for a vast target audience to root their Android devices.

[3] `https://emanuele-f.github.io/PCAPdroid/quick_start.html`

Another area, in which this project still lacks is the blocking capabilities of potentially malicious connections. This capability is provided by the Android VpnService API which is being utilized only on PCAPdroid as a paid feature. Since this project only receives the required information via a JSON (JavaScript Object Notation) message and does not have direct access to the aforementioned API, lack of blocking capabilities still persists as an inherent limitation.

1.6 Method Overview

This application as mentioned before utilizes PCAPdroid as a parent application providing the necessary data for further functionalities. PCAPdroid makes use of Android VpnService package and its corresponding APIs to passively sniff the network traffic while leaving their connections untouched. It also provides an extension to transmit the captured data in PCAP/PCAPNG format to a local/remote location for further monitoring and inspection. This feature of PCAPdroid is used in this thesis project as the main communication channel between Thread Detector and PCAPdroid. However, for this message transfer to take place, a communication channel must be established. The options provided by PCAPdroid to fullfil this need, are *HTTP Server, UDP and TCP exporters*[4]. As a result, it is made possible to receive the PCAP/PCAPNG file in the JSON format via a local TCP/UDP server.

This project makes use of TCP exporter feature of PCAPdroid and implements a local TCP server that receives and interprets the incoming packets in real-time. Subsequently, the information is parsed to a JSON interpreter and the required data including destination IP address, associated application UID, etc. are extracted that go through a preparation sequence for an inquiry from AbuseIPDB.

Finally, untill the exhaustion of daily free API calls provided by AbuseIPDB, Threat Detector inquires the maliciousness of the destination IP address for each outbound connection.

1.7 Structure of this Article

This article illustrates the process of design, implementation, working, and evaluation of Threat Detector as follows:

- **Background:** This section depicts the main idea of the application and the series of attempts that lead to some deviations from the the initial approach and some brief elaborations regarding each alteration.
- **Related:** This chapter exhibits the available inspection and monitoring solutions and evaluates each, with the goal in mind to illustrate their weaknesses and strengths in their implementation and workings. Additionally, it explains the use of a monitoring solution that is necessary for the correct functioning of this thesis project.
- **Design:** How the application is shaped around the idea and how the devised goals will be achieved are the main topics that will be cover in this chapter. It explains what sort of functionalities will be vital to fullfil the application idea and the requirements to be met.
- **Implementation:** This chapter covers the step-by-step process of implementing each of the design concepts from reception of PCAP file to destination IP address and UID extraction. Moreover, it covers how the design metrics and architecture are carried out to satisfy efficient performance, implement user-friendly UI (user interface) and

[4] `https://emanuele-f.github.io/PCAPdroid/dump_modes#25-tcp-exporter-pcap-over-ip`

comprehensive coordination with PCAPdroid. It exhibits UI components and elaborates on the back-end functionalities they execute.

- **Setup with PCAPdroid:** This section goes through the process to set up Threat Detector with PCAPdroid, enabling necessary extensions, altering some settings and finally, ensuring the adjustments are aligned with Threat Detector's configurations.
- **Evaluation:** This chapter carries out a comprehensive assessment, evaluating Threat Detector's features and functions as a final product and depicts whether this thesis's defined goals have been accomplished.
- **Conclusion:** As the final section of this paper, conclusion elaborates on Threat Detector as a product and its feasibility for large-scale use. Moreover, it mentions the difficulties and challenges this project was confronted with, which areas in real-time monitoring and threat detection look promising to explore more and which ones suffer from inherited challenges impossible to overcome with typical approaches in the context of mobile security solutions.

1.8 Contributions of this Thesis

This thesis contributes to the field of mobile and Android cybersecurity by illustrating an effective methodology for an app-level threat detection and intelligence, real-time monitoring, and also proactive risk management solution. The results and findings of this project, emphasizes both the potential and the limitations of mobile threat detection systems. It also represents a foundation for future work, research and actions in securing Android devices in our increasingly complex and hyperconnected environments.

2 Background

This Chapter provides a background and the motivation for development of Threat Detector. It outlines some of the fundamental gaps in the current mobile network monitoring and packet inspection systems and the unique challenges they faced which led to the creation on this novel approach. This chapter explains the initial idea of Threat Detector as a covering solution for the areas that other mobile monitoring systems are not presenting or strongly lacking in.

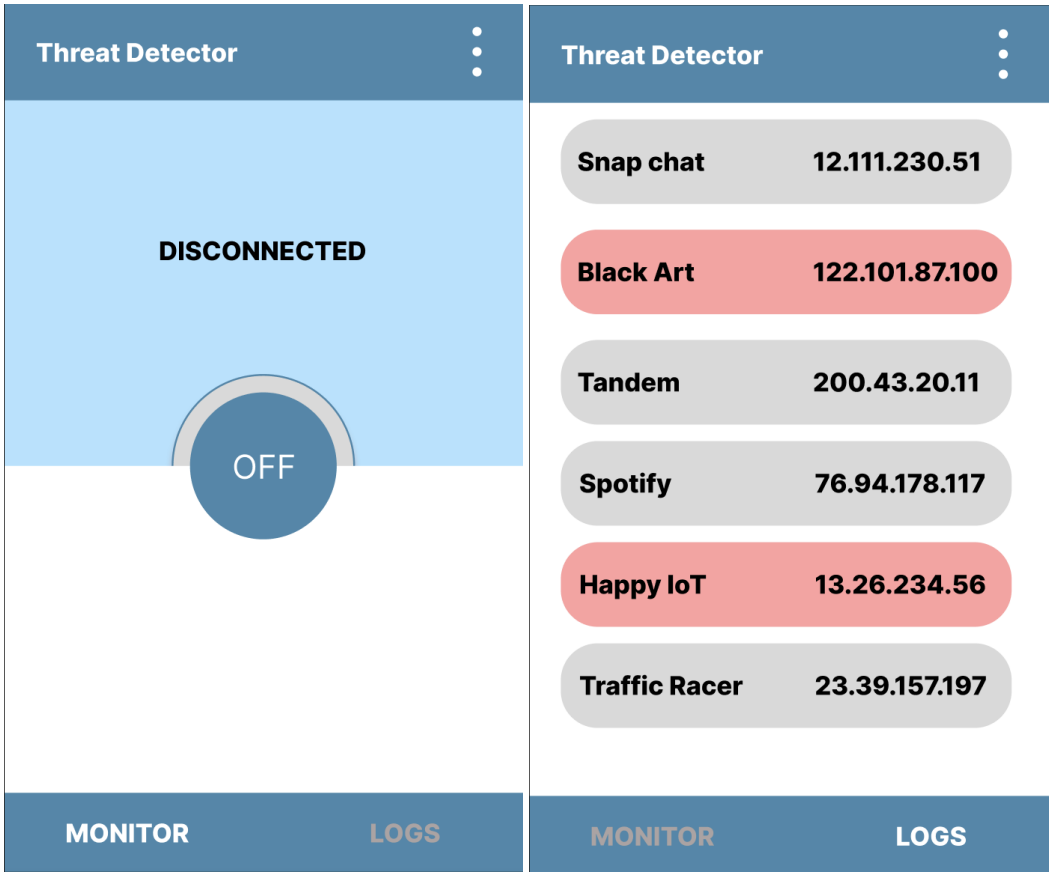
Existing solutions, while useful in some certain cases, often struggle or completely fail to address key aspects such as packet-to-application mapping, real-time threat detection, reputation checking, and user-friendly implementation mentioned features. Moreover, this chapter elaborates on the inherent difficulties in the implementation of the initial idea that make such an attempt using typical approaches close to impossible. Many of these restrictions stem from the Android operating system, limitations of the available APIs, and complexities of handling network traffic without degrading performance or user experience. Additionally, selected code snippets and API usage are presented to illustrate the workings of such systems and to highlight the motivating reasons behind certain deviations that become necessary during the course of design and development of Threat Detector.

2.1 Initial Idea

Development of Threat Detector commenced as a stand-alone idea that integrates packet interception and monitoring, cross-referencing packets and corresponding applications, along with real-time reputation check capabilities.

To achieve the aforementioned goals, Threat Detector should implement a VPN functionality to intercept network traffic while being able to route the packets back and forth between the source and destination IP addresses. Moreover, it should inspect the network traffic to find IP packet associations with each application based on their UID, and finally send a crafted inquiry to a reputation-check database such as AbuseIPDB to explore the maliciousness of device's outbound traffic while providing the user with real-time updates and notifications about any suspicious activity.

After the full development of the application based on the initial idea, Threat Detector's abstract user interface should have supposedly look like Fig. 2.1.



(a) Monitor Page (b) Logs Page

Figure 2.1: Threat Detector UI based on initial idea, Own Creation

2.1.1 VpnService API - Android's VPN Functionality

The APIs provided by Android as an OS (Operating System) include a wide range that includes functions to handle network traffic based on the user context (e.g. normal or root). According to the Android's *official API website*[5], the preferred approach towards network traffic interception is utilization of VpnService API. This API provides an active interception of network and internet traffic by routing the packets through a gateway (normally the default gateway of the device's NIC (Network Interface Card)). This implementation ensures that the commute of network packets takes place only through one communication path. As a result, inspection of network traffic can be carried out only on that gateway simplifying packet handling and re-routing.

As shown in the following code snippet provided on Android *developer website*[6], Android system establishes a TUN (Tunnel) interface to route all the packets utilizing a VpnService builder and routes them though the localhost address.

[5] `https://developer.Android.com/reference/android/net/VpnService`
[6] `https://developer.Android.com/develop/connectivity/vpn`

```
// Configure a new interface from our VpnService instance. This must be done
// from inside a VpnService.
val builder = Builder()

// Create a local TUN interface using predetermined addresses. In your app,
// you typically use values returned from the VPN gateway during handshaking.
val localTunnel = builder
        .addAddress("192.168.2.2", 24)
        .addRoute("0.0.0.0", 0)
        .addDnsServer("192.168.1.1")
        .establish()
```

Figure 2.2: VpnService Builder Implementation (Source: [1])

Using this VPN the user will be able to:

- Read raw packets going through the established Virtual Private Network.
- Analyse or filter them.
- Apply inbound and outbound rules.
- Inject data into the commuting traffic.

In simpler terms, it acts as a virtual network adapter that Android routes all the traffic through. This enables interception, monitoring, and packet modification before forwarding to its original destination.

Here is a partial depiction of VpnService implementation in the initial idea:

```
class AppVpnService (): VpnService() {
    private var vpnInterface: ParcelFileDescriptor? = null
    private var running = false
    private val vpnScope = CoroutineScope(Dispatchers.IO + SupervisorJob())

    @RequiresApi(Build.VERSION_CODES.VANILLA_ICE_CREAM)
    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {

        if (intent?.action == "STOP_VPN") {
            Log.d("AppVpnService", "Received STOP_VPN action")
            stopSelf() // <-- This will now trigger onDestroy()
            vpnInterface?.close()
            vpnInterface = null
            return START_NOT_STICKY
        }

        if (running) return START_STICKY
        running = true

        val builder = Builder()
        builder.setSession("AppVpnService")
            .addAddress("10.0.0.2", 32)
            .addDnsServer("8.8.8.8")
            .addRoute("0.0.0.0", 0)
            .setBlocking(true)
```

Figure 2.3: Defining Default Gateway, Own Creation

```kotlin
vpnScope.launch {
    vpnInterface?.fileDescriptor?.let { fd ->
        val input = FileInputStream(fd)
        val channel = input.channel
        val buffer = ByteBuffer.allocate(32767)

        try {
            while (running) {
                buffer.clear()
                val readBytes = channel.read(buffer)
                if (readBytes > 0) {
                    buffer.flip()
                    parsePacket(buffer)
                }
            }
        } catch (e: Exception) {
            Log.e("AppVpnService", "VPN read error: ${e.message}")
        } finally {
            input.close()
        }
    }
```

Figure 2.4: Reading Incoming/Outgoing Data, Own Creation

In the screenshots provided in Fig. 2.3 and Fig. 2.4, you can see the VpnService builder implementation that is responsible to route all the traffic to the default gateway alongside a ParcelFileDescriptor. This is an Android class that acts as a wrapper around a Linux file descriptor. A file descriptor is a handle used by the operating system to access files, sockets, pipes, and I/O resources. However, luckily Android provides a high-level, safe, Kotlin/-Java friendly version of that through ParcelFileDescriptor API. In this implementation, the ParcelFileDescriptor instance represents a virtual network interface created by VpnService to handle the transfer of packets through tunnelling.

```kotlin
@RequiresApi(Build.VERSION_CODES.VANILLA_ICE_CREAM)
private suspend fun parsePacket(buffer: ByteBuffer) {
    buffer.order(ByteOrder.BIG_ENDIAN)

    val version = (buffer.get(0).toInt() shr 4) and 0xF
    if (version == 4 && buffer.limit() >= 20) {
        val destIp = "${buffer.get(16).toInt() and 0xFF}."
                "${buffer.get(17).toInt() and 0xFF}." +
                "${buffer.get(18).toInt() and 0xFF}." +
                "${buffer.get(19).toInt() and 0xFF}"
```

Figure 2.5: Extracting Destination IP Address from Raw Packets, Own Creation

Fig. 2.5 depicts the extraction of destination IPv4 address from a raw IP packet header. This IP address will be parsed to a threat intelligence database for an inquiry regarding the maliciousness of the address.

2.1.2 Application Name and Icon Extraction

As depicted in screenshots of the initial idea, the inspected application name and possibly the its icon should be illustrated for a more user-friendly exhibition of outbound network connections and their originating applications.

Intercepted packets using Android's VpnService API only provide source/destination IP address, ports, protocols, and payload data (normally encrypted via SSL/TLS). Thus, retrieving which application is responsible for the out-going transmission of a specific packet is not feasible.

In order to extract the application's name, the packet's UID (Linux user identifier) should be extracted. Subsequently, it should be mapped to the installed application package. For Android is a Linux-based system, to access the UID required to extract the app-generated traffic, retrieving the content of tcp/udp virtual folder is necessary. The files are located in `/proc/net/tcp` or `/proc/net/udp`. These files map network sockets to UIDs. Once the UID is extracted, it can be resolved to retrieve the application name via Android's PackageManager API.

Simplified illustration:

```
Packet -> IP/Port -> /proc/net/* -> UID -> Package name -> App name
```

2.1.3 Limitations and Problems

The above-mentioned employment of VpnService alongside extraction of packet UIDs fullfils the requirements of the initial concept. However, implementing a VPN through VpnService API and extracting packets' UIDs introduce several challenges which necessitates some modifications to the original idea.

Routing Problem

The way that a VPN service functions is to create a connection point between multiple networks and enables the transmission and reception of packets among systems within them. However, Android's VpnService does not provide an automatic mechanism to redirect the incoming and outgoing packets.

The referenced implementation of VpnService from the official developer website also lacks the APIs that comply with guidelines vital for correct functionality of a VPN. As a result, additional modifications are needed to handle the packet redirection in a VPN system, designed on top of VpnService.

In the most VPN protocols including IPsec, network traffic is transmitted using encapsulated IP packets. This means an IP packet is placed inside another one which results in two distinct headers. Inner and outer headers. Inner header contains the original IP header and the payload, while the outer header is added for transport across the network and is removed once the packet reaches its destination.

A fully functional VPN that meets all the requirements of this approach needs to determine a default gateway, route all the traffic through it, transmit packets across network using IP packet encapsulation, and finally use other Android APIs such as streams to correctly handle the incoming and outgoing TCP/UDP streams while ensuring efficient buffering.

Unfortunately, since Android does not provide the APIs for complete header management and packet redirection across network, its implementation requires all of the these steps to be carefully handled at the application level. Consequently, the utilization of Android VpnService API for building a fully functional VPN is highly impractical in this project, as its implementation deviates drastically from the original scope and objectives of this thesis.

Application UID Extraction Problem

As stated before, to determine the application name, from which the outbound internet packet originates, extracting the UID of the packet is necessary, which is only accessible via `/proc` filesystem.

Accessing `/proc` filesystem is possible via the following approaches:

- Having root access.
- Utilizing system-level permissions.
- Using native C programming to interface directly with the Linux kernel of the Android device.

From the above-mentioned options, none of them is feasible to use. Rooting the Android device for a normal user brings along a lot of drawbacks including weakened security and extended susceptibility to various attack surfaces. Moreover, as of Android version 11 and above, the system-level permissions are not viable. As a result, this approach works only on Android Q and below.

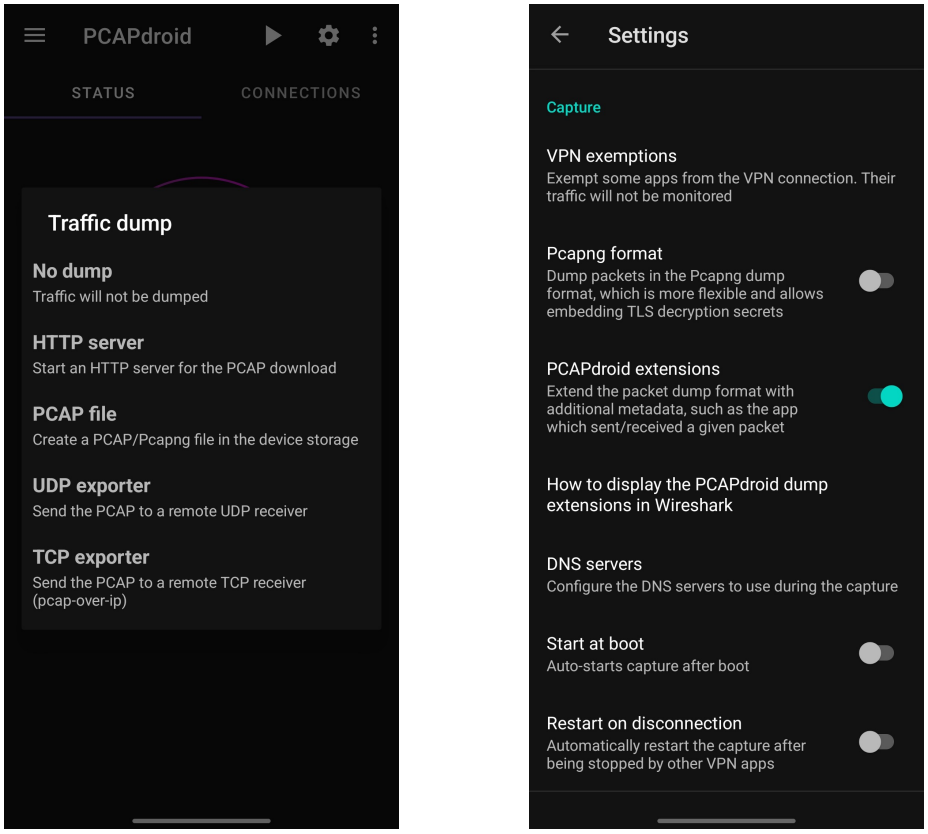
Interfacing the Linux kernel using C language remains the only reliable and feasible option that network monitoring solutions such as PCAPdroid and Ant-Monitor employ. Implementing this however, is not feasible for this thesis.

2.2 How PCAPdroid already does it all

Developing a network monitoring solution similar to *Wireshark*[7] for mobile platforms has already been a topic of importance among enterprise personal by the emergence of Bring Your Own Device policy. As a result, applications such as PCAPdroid, Mopeye, and Ant-Monitor came to existence as a solution to give administrators some leverage examining the network activities.

Among the aforementioned solutions PCAPdroid is an open-source solution that has implemented a fully functional VPN along with an extensive interface with Android's Linux kernel to thoroughly handle `/proc` filesystem. Thus, PCAPdroid can be a suitable candidate to provide the underlying foundation for the correct functionality of Threat Detector. It has settings that can be adjusted for communication between the Android device and the administration endpoint. It captures network traffic and stacks the information in a widely-known PCAP/PCAPNG formats that can be fed into other network inspection tools such as Wireshark though various communication methods. These methods include HTTP servers and TCP/UDP streams.

[7] `https://www.wireshark.org/`



(a) PCAPdroid Traffic Dump Options (b) PCAPdroid Packet Capture Settings

Figure 2.6: Usage of VpnService API, Self Taken

As shown in the screenshots above with employment of TCP exporter functionality and activation of PCAPdroid extensions, majority of the limitations and inherent restrictions that stem from Android's ecosystem and architecture can be overcome. The TCP exporter enables information transfer using TCP protocol and PCAPdroid extensions provides the application package name that can be easily utilized to extract the application name and icon necessary to meet the original idea of this thesis project.

2.3 Integration with PCAPDroid

As explained, for Threat Detector to function properly and for the requirements of this thesis to be met, integration/implementation of an approach to route and intercept network traffic and access to `/proc` filesystem are of utmost importance. However, since a thorough execution of them is unfeasible for this project, integration of those functionalities through a communication medium can potentially be a proper solution.

The following chapters provide a detailed discussion of the coordination between PCAPdroid and Threat Detector. In this setup, data is exchanged in the form of PCAP files via a local TCP server, thereby enabling Threat Detector to address functionalities that would otherwise be impractical to implement on its own.

3 Related Work

In the recent years, as indicated in previous chapters, the reliance on mobile devices has significantly increased within enterprises. However, usage of mobile devices is not limited for enterprise networks and it plays a huge role as an inevitable contributor to our daily lives. Therefore, it is of utmost importance not only for enterprises but also individuals to be able to ensure the security of their mobile devices. As a result, a wide range of network monitoring and inspection applications have been developed to enable researchers to capture, inspect, and also analyse network traffic generated by mobile applications.

Network monitoring and inspection solutions are essential to detect suspicious communication patterns, privacy leakages, and network and application-related anomalies that may compromise user data and the enterprise's system integrity.

Among a large number of prominent solutions developed in this area, **PCAPdroid, Mopeye**, and **AntMonitor** have illustrated notable progress towards enhancing transparency along with control over mobile network traffic. PCAPdroid provides a non-root approach towards packet capture and employs Android's VpnService API to intercept and record traffic in PCAP/PCAPNG format allowing comprehensive network inspection without the need for elevated privileges. On the other hand, Mopeye aims its focus towards large-scale network measurements and data aggregation for further research and monitoring purposes. It considerably contributes to mobile network performance and security. AntMonitor however, takes a privacy-oriented approach by enabling real-time analysis of application traffic, providing thorough visibility into how user data is transmitted over a network.

These existing tools have constructively contributed to the advancement of mobile traffic analysis. However, each of them also presents limitations in terms of integration, flexibility, and real-time threat intelligence and detection. The following sections provide an overview in details, highlighting their architectures, capabilities along with their constraints, which consequently became the motivating factors for Threat Detector development.

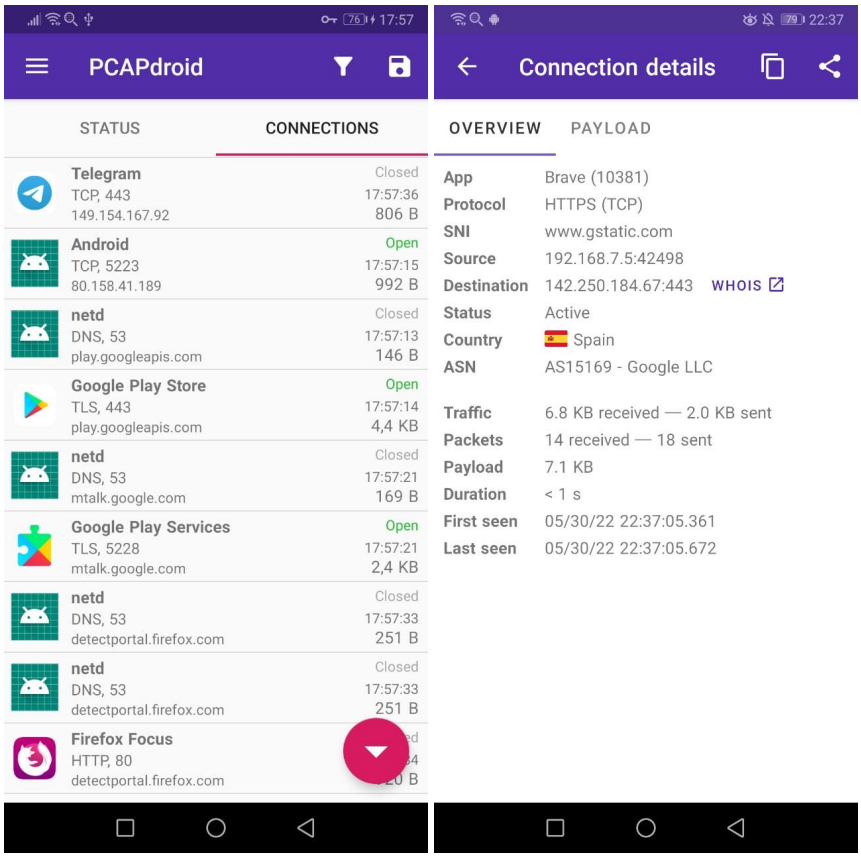
3.1 PCAPdroid

As one of the most recognized open-source Android applications for capturing and analysing network traffic, PCAPdroid operates entirely without root privileges by leveraging Android's VpnService API. This results in the redirection of all network packets through a local virtual interface, allowing PCAPdroid to passively sniff and monitor network traffic. Moreover, it can log and export captured network traffic in the form of

PCAP/PCAPNG files; consequently, making it compatible with standard packet analysis tools such as Wireshark or *tshark*[8].

When it comes to PCAPdroid's architecture, it establishes a user-space VPN that intercepts both TCP and UDP packets. Once these packets are captured they are parsed locally or can be transferred to a remote system for further analysis as data streams in real-time. This is the very characteristic of PCAPdroid that makes it an efficient bridge between on-device data collection and remote analysis systems. This design enables PCAPdroid to include support for packet filtering, payload decoding, and also per-application traffic mapping. This, as mentioned before, takes place by correlating sockets with applications' UIDs.

PCAPdroid's primary advantage lies within its non-root nature, making it an attractive option for users, enterprises, and researches who cannot or do not wish for a drastic modification in their device operating system. Worth to mention that mission of such applications is to assist securing mobile devices and rooting or jail-breaking removes the security measures implemented on those devices, making them susceptible and puts them in significant risk of attacks. While providing a functional application without rooting the device, PCAPdroid also maintains the balance between technical depth by visualizing live traffic statistics and sustaining the compatibility with standard PCAP/PCAPNG workflow. These can be observed in the screenshot of PCAPdroid's settings in the previous chapter and also in Fig. 3.1.



(a) PCAPdroid Connections Page (b) PCAPdroid Connection Details Page

Figure 3.1: Screenshots from PCAPdroid [2]

[8] https://tshark.dev/

While providing considerable advantages and the foundation for other network analysis tools to build upon, PCAPdroid was not originally designed for threat detection and intrusion analysis. Even though it captures traffic effectively, it does not incorporate mechanism to identify malicious patterns, command-and-control connection attempts, and data exfiltration. Moreover, the VPN-based model implemented in PCAPdroid introduces a minor performance overhead that stems from its interception model. This may cause it to interfere with some network applications that follow strict network policies and use encrypted VPN tunnelling techniques.

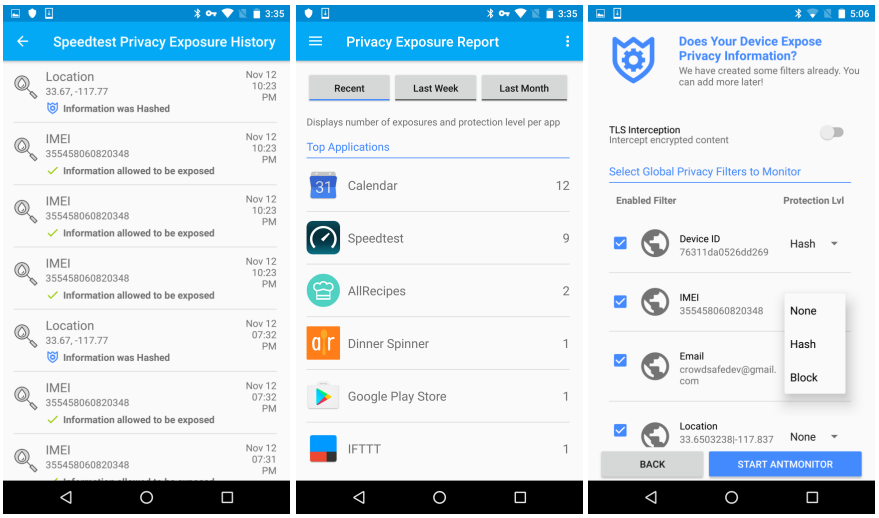
Despite the limitations, PCAPdroid remains a solid tool for Android network traffic analysis. Its modular design that follows PCAP/PCAPNG formatting alongside being an ideal foundation for other solutions to implement its open-source core, make it a reliable underlying base for other applications such as Thread Detector to extend its functionality.

3.2 AntMonitor

While introduced as an alternative for PCAPdroid on its *GitHub page*[9], AntMonitor represents a slightly different approach towards mobile traffic monitoring and analysis. Its focus lies mainly on privacy protection and transparency for end users. Developed by researchers at the University of California, AntMonitor was developed both as a research platform and privacy-enhancing tool. The goal it aims to achieve is to assist users understanding the workings behind how an application sends personal data across networks, in particular to third-party domains and analytic services.

Similar to PCAPdroid, AntMonitor heavily relies on Android's VpnService API to capture network packets. Moreover, it represents a more privacy-aware architecture, implementing modules covering real-time packet analysis. These modules classify network flows, detect leakage of sensible information such as Personally-Identifiable-Information (PII), and also visualize the communication between a device and a remote server.

In Fig. 3.2 you can see the privacy-awareness implementation.



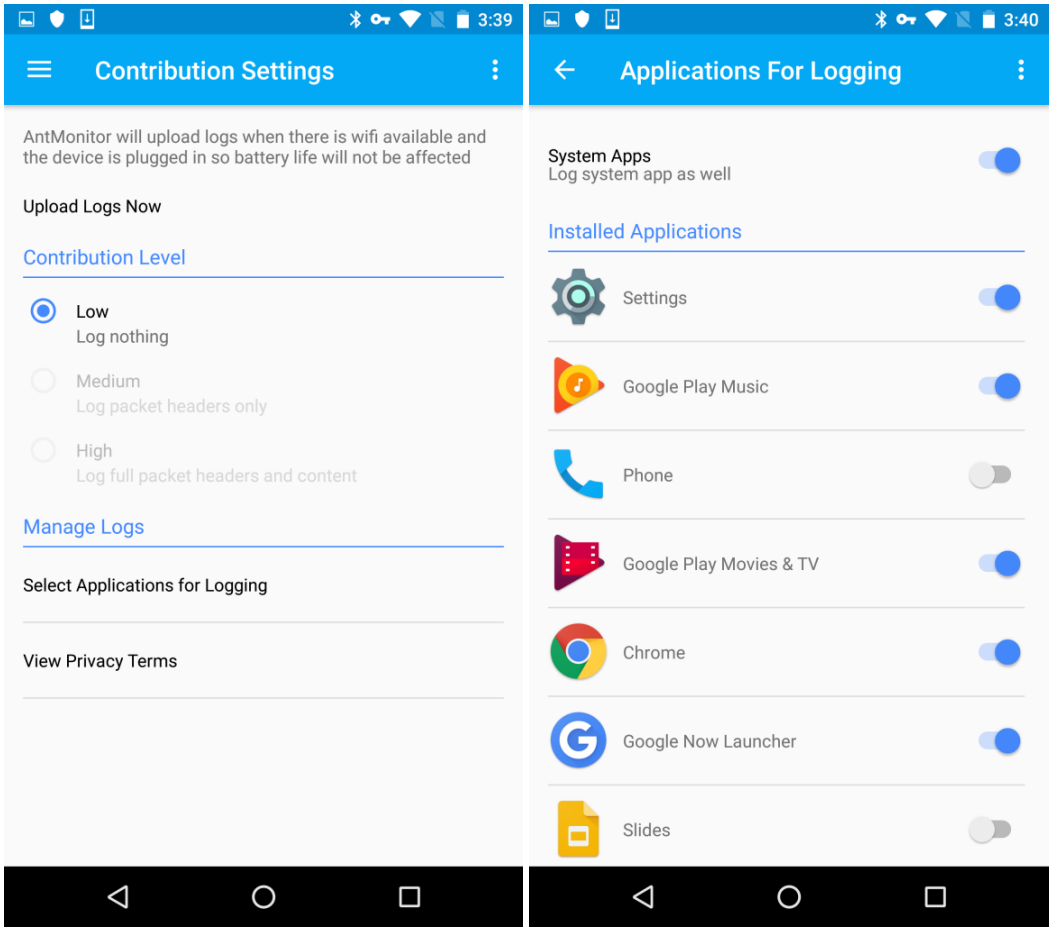
(a) Privacy Exposure History (b) Privacy Exposure Report (c) Privacy Exposure Information

Figure 3.2: Screenshots from AntMonitor Privacy-aware Architecture [3]

[9] https://github.com/UCI-Networking-Group/AntMonitor

AntMonitor also features a framework for server-side traffic analysis. It accumulates anonymized data from multiple devices to build global statistics about mobile data leakages and application behaviours. This can be considered as a hybrid approach to combine local inspection with cloud-based aggregation, allowing scaling of modern mobile privacy studies.

In Fig. 3.3 you can see the aggregation options regarding anonymized logs and the applications contributing to them.



(a) AntMonitor Log Contribution Settings (b) AntMonitor Applications for Logging

Figure 3.3: Screenshots from AntMonitor Logging Capabilities [3]

One of AntMonitor's key strengths is its focus on privacy awareness of end users. It provides the user with detailed insights about what sorts of information an application is transmitting and subsequently assists the user to make informed decisions about permissions and trustworthiness of that application. This privacy-centric approach however, introduces some notable trade-offs. In this design, data protection and user consent are prioritized, which limits AntMonitor's applicability for low-level packet inspection and complicates security research that require intense packet analysis. Additionally, AntMonitor's architecture is not designed with real-time threat detection and IP blocking capabilities in mind. This makes it less suitable for use cases involving direct intrusion detection.

As a brief summary, AntMonitor depicts a significant advancement in privacy-preserving traffic analysis but suffers from the lack of mechanisms to prevent active threats. Nonethe-

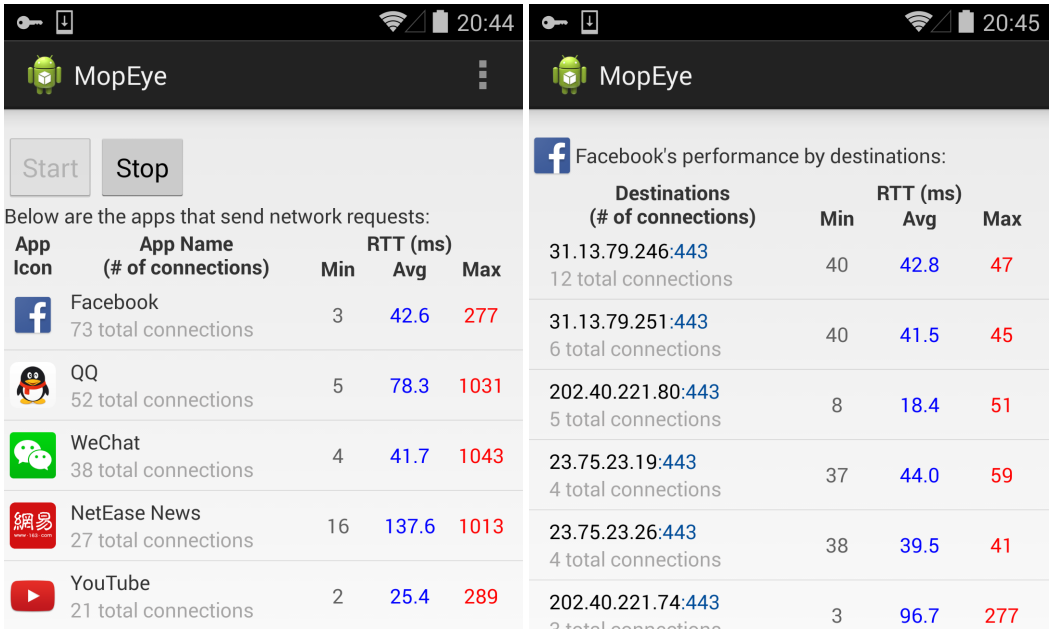
less, the principles employed in its design such as on-device analysis for mobile platforms and user transparency, have subsequently inspired the development of Threat Detector, which combines security analytics along with privacy awareness.

3.3 Mopeye

Another research-oriented attempt aimed at large-scale mobile network monitoring is *Mopeye*[10][4]. Unlike AntMonitor and PCAPdroid that focus on applications for per-device traffic inspection, Mopeye performs as a distributed network measurement platform that enables researchers to collect data across various mobile devices within different networks.

The mission it aims to accomplish is to understand the performance, security characteristics, and reliability of mobile networks, allowing participants to carry out experiments that measure parameters including packet latency, loss throughput, and DNS query behaviour. It is also known to be a metadata-collecting platform aggregating data about network configurations and signal strength which are vital for assessing real world network conditions.

Fig. 3.4 illustrates Mopeye's functionality.



(a) Mopeye Network Performance page (b) Mopeye Application Performance page

Figure 3.4: Screenshots from Mopeye [5]

From a technical viewpoint, Mobeye implements a client-server architecture, in which mobile clients perform network measurements, followed by a report that is sent to centralized servers. It provides APIs for task scheduling, result collection and its visualization. Moreover, Mobeye is designed in a way that makes it suitable for crowdsourced network monitoring and therefore, has been used in various studies concerning mobile security and performance anomaly detection.

[10] https://www4.comp.polyu.edu.hk/~csrchang/MopEye.pdf

However, as a solution primarily focused on macroscopic network traffic analysis, it does not provide packet-level inspection. It neither captures/inspects individual packet flows, nor does it identify suspicious or privacy-violating communication flows. In spite of the fact that it offers valuable insights about the overall network health, its capabilities are undoubtedly limited when it comes to device-level security and malware detection. However, despite aforementioned constraints, Mopeye provides a strong underlying foundation in understanding the broader mobile network context, in which they operate. Mopeye complements other monitoring tools such as PCAPdroid and AntMonitor that inspect network traffic on a microscopic level. Insights and network metadata provided by Mopeye, can inform threat detection models by supplying background data about normal network performance and connectivity patterns.

3.4 Comparison

Even though PCAPdroid, AntMonitor, and Mopeye share common goals to achieve, increasing visibility of mobile network traffic, their objectives and design philosophies are substantially different.

In the following table you can observe their functionalities and subsequent differences.

Table 3.1: Comparison of Existing Mobile Network Monitoring Tools, Own Creation

Tool	Main Purpose	Requires Root	Focus	Real-Time Detection	Privacy Protection	Data Format
PCAPdroid	Packet capture and export	No	Network analysis	No	Moderate	PCAP
AntMonitor	Privacy-aware traffic inspection	No	Data leakage and app transparency	Partial	Strong	Internal database
Mobeye	Network measurement and performance	No	Large-scale monitoring	No	Limited	Aggregated metrics

As illustrated in table 3.1, PCAPdroid excels in network visibility on packet-level, making it a suitable option for forensic or research-centric traffic analysis. On the other hand however, it lacks intelligence for automatic threat detection and classification. AntMonitor brings along useful user privacy insights but does not incorporate extensive traffic inspection, neither does it support automatic response mechanisms. As a contributor to macroscopic network analytics, Mopeye provides complementary information about the overall network condition but does not highlight application-specific data.

In contrast however, Threat Detector developed in this thesis aims to accomplish what the above-mentioned solutions lack in their implementation. It seeks to bridge the gap between passive network traffic monitoring and active threat detection. By integrating the packet-capture capabilities provided by PCAPdroid with additional intelligence modules to query external threat intelligence sources (e.g. AbuseIPDB), Threat Detector improves security awareness on the device. The integration of PCAPdroid and Threat Detector enables real-time detection of suspicious IPs, provides mapping functionality to associate network packets with their originating application, addressing some of the limitations observed in existing tools.

3.5 Summary

This chapter represented an overview of research efforts and their subsequent results in regard to mobile traffic monitoring and network packet analysis. PCAPdroid supplies a robust, open-source approach towards network packet and application level analysis while eliminating the need for rooting the device. AntMonitor highlights the user privacy as its primary mission, while Mopeye contributes to large-scale network performance analysis.

While each of the mentioned tools has advanced the field of mobile security in particular ways, their combined limitations including lack of real-time threat detection, absence of application/IP blocking, and restricted integration capabilities underline the need for a more comprehensive solution. Threat Detector is built upon their foundation and aims to provide a hybrid model, integrating real-time threat detection, application-to-IP mapping, alongside flexibility to be interoperable with existing tools such as PCAPdroid.

The following chapters in this paper elaborate on the implementation of Threat Detector, explaining how its architectrual design overcomes the identified gaps in the existing tools.

4 Architecture/System Design

4.1 Overview

This chapter represents the design architecture of the proposed mobile network monitoring system, Threat Detector. This application aims to detect the potentially malicious network traffic on Android devices with a passive packet sniffing approach. It is developed as an enhancement and an extension for the open-source applicaion PCAPdroid, integrating additional modules for IP reputation checking and application-to-ip mapping, while using the state-of-the-art user interface and user experience implementations to meet the design expectations of a modern Android application.

As covered further in this chapter, Overview elaborates on system components, depicting a modular approach that is employed to develop Threat Detector. It is then followed by some illustrations, explaining the application's function though the use of a flow diagram. Finally, the reasons behind those decisions are summarized.

Application's architecture is designed based on three main objectives:

- **Transparency:** Intercepting and logging of network traffic without altering or blocking the normal traffic flow.
- **Association:** Accurately associating each network connection with its originating application.
- **Threat Detection:** Assessing the reputation of the destination IP address with the assistance of external IP intelligence sources such as AbuseIPDB.

The overall structure of Threat Detector follows a modular design which ensures separation between packet capture, user interface, and data processing.

As shown in Fig. 4.1, the first layer is handled by the Android operating system which provides access to the applications' network traffic. Second layer is implemented by PCAPdroid that forwards all of the sniffed packets to a local gateway for centralized analysis. As the final layer that PCAPdroid is responsible for, data assembly layer builds the PCAP/PCAPNG files and transmits them to a local server using TCP. Subsequently, Threat Detector analyses the received data, queries AbuseIPDB for IP reputation, and notifies the user.

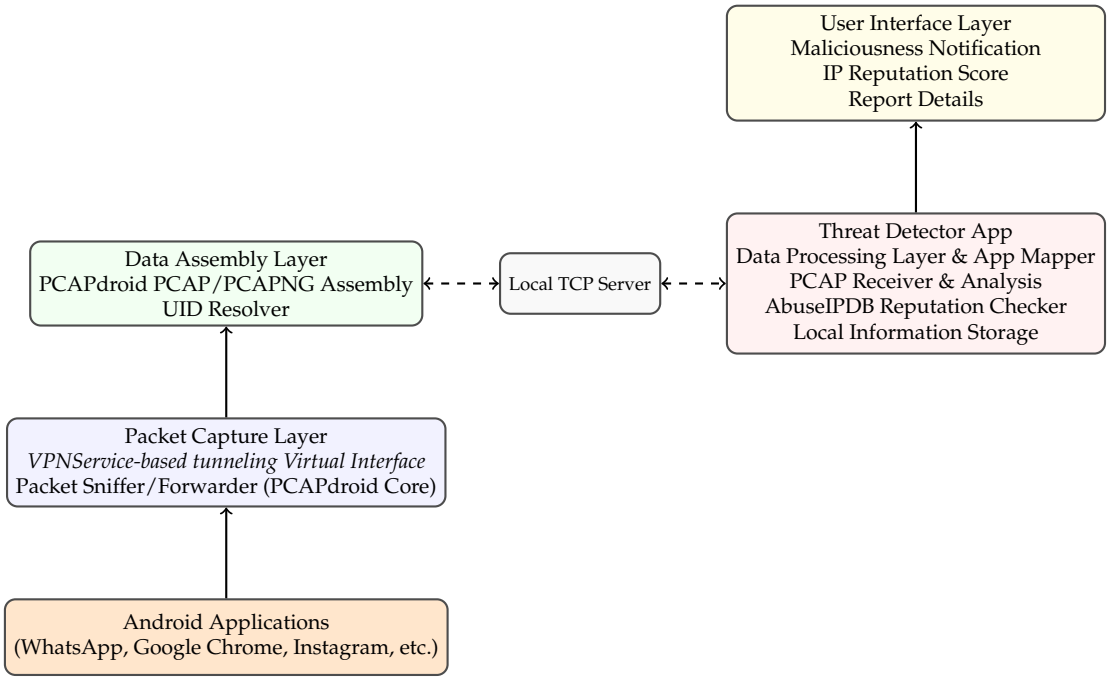


Figure 4.1: Modular System Architecture of Threat Detector, Own Creation

4.2 System Components

Threat Detector as a systems is composed of several independent stand-alone components responsible for a seamless functionality that follow the "Separation of Concerns" model for ease of debugging and further development.

4.2.1 VPN-based packet Capture Module

This module establishes a virtual private network interface (through utilization of PCAPdroid) that captures the incoming and outgoing packets from user's applications without requiring root access on the Android device. Raw captured packets are then immediately forwarded to the next module for further analysis while keeping the communications between endpoints alive.

The functions of this module are as follows:

- Initialization and maintenance of a virtual interface (tunnel interface).
- Capturing raw packets from the device's network traffic while preventing the modification of the packets.
- Forwarding the packets to the local processing engine of PCAPdroid.

4.2.2 UID Resolver and Application Mapper

This module takes the responsibility of correlating raw captured packets with the originating Android applications. Each packet commuting through the network has a retrievable

UID that can be extracted using system calls. As the last step, this module employs Android's *PackageManager*[11] to map the UIDs to human-readable application package names.

4.2.3 AbuseIPDB Reputation Checker

This component integrates the use of external threat intelligence sources such as AbuseIPDB. For each of the unique IP addresses detected, it queries a reputation check determining whether the IP address has been reported as malicious/suspicious.

Functions of this module are as follows:

- Sending HTTP requests to AbuseIPDB using its API querying IP's maliciousness.
- Parsing the JSON response which contains IP's confidence score and report details.
- Caching results for higher performance.
- Communicating reputation data with the UI layer for user depiction.

This module comes with some the following design considerations:

- Efficient querying avoiding redundant API calls.
- Asynchronous execution of reputation checks to prevent UI blocking using an additional thread.
- Respecting API rates and response interpretation.

4.2.4 Inter-Application Communication

This module enables an enhanced architecture to include a local TCP servers for the communication between PCAPdroid and Threat Detector to take place. The captured data is then transferred between the two in a form of PCAP/PCAPNG file.

Functions of this module are as follows:

- Creation and establishment of a TCP socket server within Threat Detector.
- Transmission of captured packets from PCAPdroid to Threat Detector.
- Enabling Threat Detector to parse, analyse, and visualize data for user.

The aforementioned design ensure modularity and scalability. It allows Threat Detector to evolve independently while receiving live traffic information from PCAPdroid.

4.2.5 Data Storage Agent and User Interface Layer

This module is not only responsible to cache the most queried IP addresses in the application's RAM, but also stores the API key and user's data in *SharedPreferences*[12].

User Interface

Threat Detector's user interface presents network and threat information in an organized intuitive format. It allows the user to view ongoing connections associated with the applications, inspect detailed IP-related information including confidence score and a report explaining the reason behind the IP being flagged as malicious.

[11] `https://developer.Android.com/reference/Android/content/pm/PackageManager`
[12] `https://developer.Android.com/training/data-storage/shared-preferences`

4.3 Data Flow and Module Interaction

The data flow (Fig. 4.2) follows a clear path from packet capture in PCAPdroid to threat assessment in Threat Detector as depicted below.

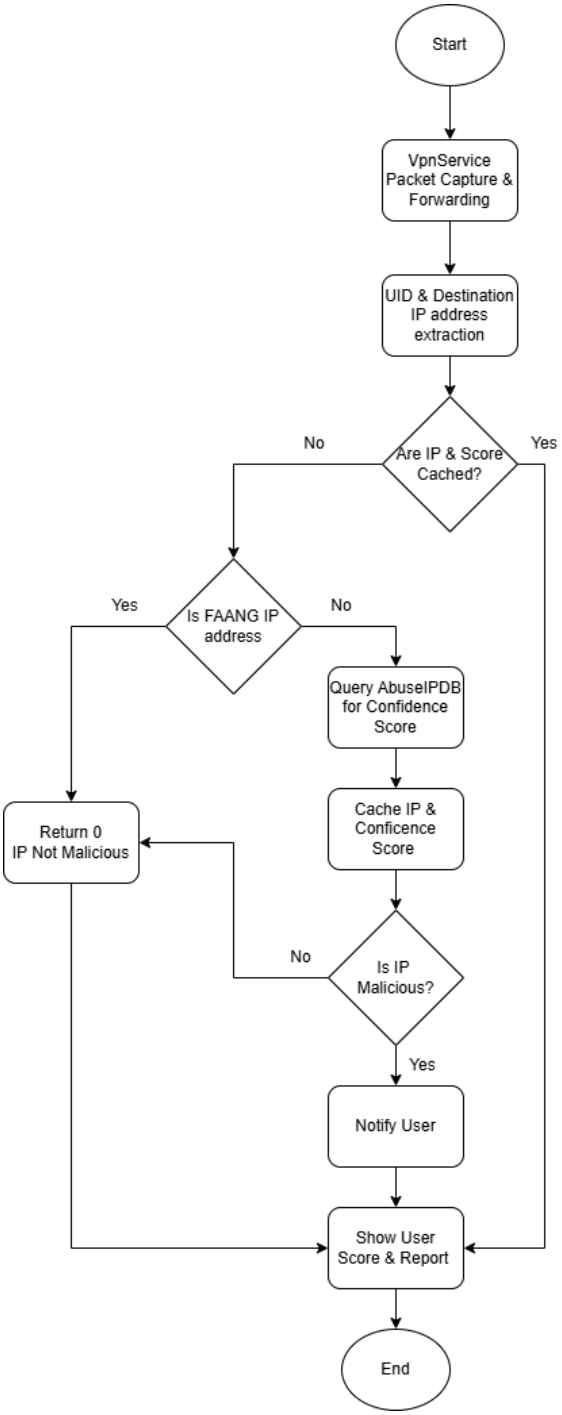


Figure 4.2: Threat Detector Flow Diagram, Own Creation

The implementation this project follows to fulfill threat assessment, can be explained by steps below:

- **VpnService Capture:** Traffic from all of the applications are accumulated and redirected to the virtual interface; in this case default gateway.
- **Packet Capture and Forwarding:** Packets are handed over to PCAPdroid processor for UID and destination IP address extraction, followed by mapping to the originating application.
- **Reputation Check:** Destination IP addresses are queried against AbuseIPDB.
- **Score Caching:** Destination IP address and its associated confidence score are stored in a map for faster future access and minimization of API calls.
- **User Notification and Info Depiction:** The confidence score, IP report, and its originating application are depicted to the user. Afterwards, the score is checked against a minimum user-set value that consequently notifies the user if it is higher than the set value.

4.4 Design Decisions

Various design choices were made throughout this project to achieve performance and modularity alongside scalability.

Table 4.1: Design Decisions and Their Justifications, Own Creation

Design Decision	Design Justification
Use of `VPNService` instead of root access	Provides a passive method for traffic interception without requiring elevated privileges, which ensures compatibility with non-rooted devices while preserving user privacy.
Extension of PCAPdroid framework	Leverages an existing open-source foundation for packet capture and logging, allowing focus to be only on threat intelligence rather than reimplementation of capture logic.
Integration of a local TCP server	Ensures modular communication through a lightweight socket connection, enabling independent development and scaling of the Threat Detector application as a stand-alone solution.
Asynchronous API calls for AbuseIPDB queries	Prevents blocking of the user interface thread and maintains responsiveness during IP reputation lookups.
Caching of Report results	Reduces redundant requests, achieving optimization and reduction of API usage and faster access for lookups.
Modular and layered architecture	Improves maintainability, scalability, and fault isolation, allowing each component (capture, mapping, reputation check, and UI) to function independently.
User Notification	Keeps the user or administrator informed about potential malicious activities with the use of real-time notification system that includes application name and its IP address.

4.5 Summary

The design architecture presented in this chapter provides a reliable foundation for packet capture, application network monitoring, and threat detection on Android devices.

Threat Detector's modular design facilitates easy updates and further development to implement additional APIs and features. Moreover, the use of VpnService ensures compatibility with various Android versions while preventing the need for root access when it comes to packet interception and application UID extraction.

5 Implementation

5.1 Overview

This chapter explains the practical implementation of Threat Detector, which is a functionality extension of the open-source application PCAPdroid. The main objective of this implementation was to design a passive network monitoring solution that is capable of identifying the source application and the destination IP address of outbound internet packets, followed by an assessment of the potential risks those connections might bring along using AbuseIPDB as a threat intelligence platform.

The practical implementation of Threat Detector follows a modular architecture that separates the responsibilities of packet capture and processing, data retrieval, and visualization. Unlike PCAPdroid and similar solutions that focus on low-level packet capture and forwarding, Threat Detector does not implement its own VPN-based interception. Instead of doing so, it operates as a server application for PCAPdroid as its client through a local TCP server, which receives real-time information in JSON format. This design and the subsequent implementation ensures Threat Detector remains lightweight and modular while keeping the focusness on data retrieval and presentation of network intelligence rather than traffic interception and manipulation.

The following sections describe the Threat Detector's system architecture, its internal components, and the integration with AbuseIPDB alongside the communication interface with PCAPdroid.

5.2 System Architecture

Threat Detector's system architecture consists of four main layers:

1. **Traffic capture layer** which is handled by PCAPdroid using Android's VpnService API to capture network traffic and subsequently build a PCAP file based upon the captured information.
2. **Data transport layer**, handled by Threat Detector which is responsible for transmission of the captured data from PCAPdroid to Threat Detector using a local TCP socket.
3. **Threat analysis layer**, implemented in Threat Detector which parses the data, queries AbuseIPDB, and assesses the reputation score.

4. **Presentation layer** that is the graphical user interface responsible for depiction of the results and configuration options to the user/administrator.

In this constellation, PCAPdroid acts as a data aggregator, while Threat Detector functions as a data processing and visualization unit.

figure Fig. 5.1 shows this flow:

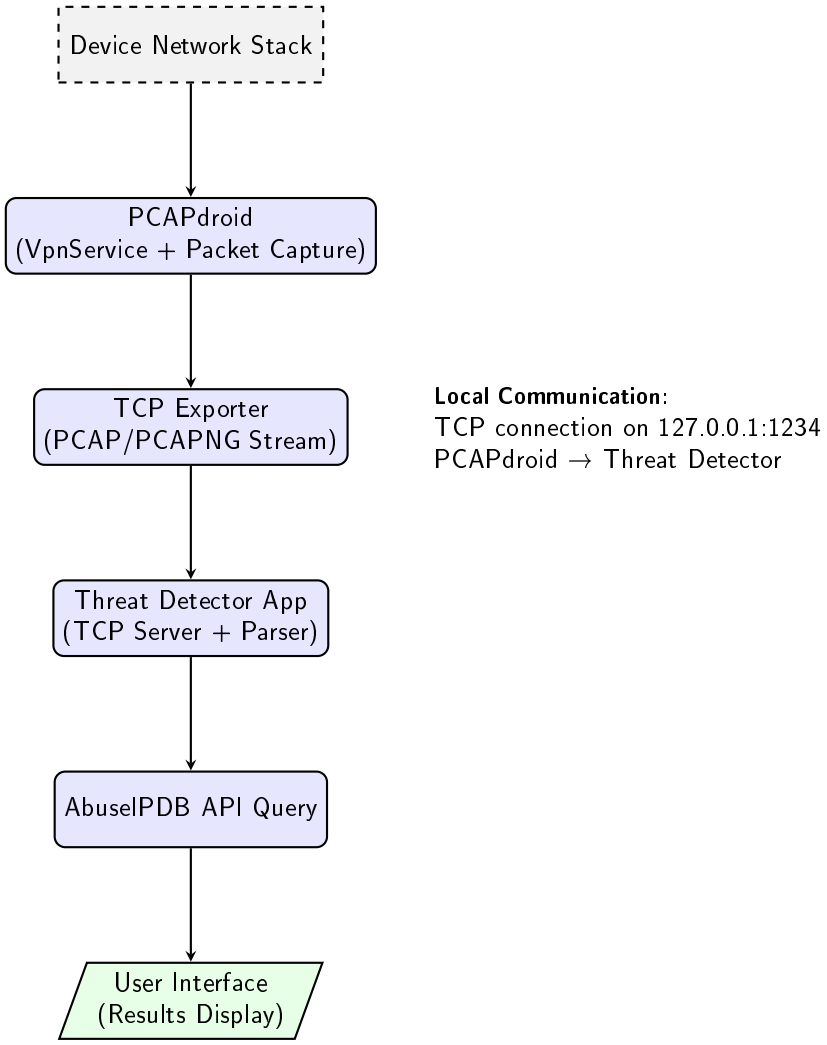


Figure 5.1: Overall system architecture showing the data flow from packet capture by PCAPdroid to IP reputation analysis by Threat Detector.

5.3 Development Environment and Setup

The implementation and the development of Threat Detector was carried out using Android Studio(Narwhal 3 Feature Drop | 2025.1.3) with target SDK (Software Development Kit) version 34 in mind. This project is programmed primarily in Kotlin providing modern language features and improved coroutine support for better asynchronous network operations. Moreover, this implementation also utilises Android's *Jetpack Compose*[13] which is

[13] `https://developer.android.com/compose`

the state-of-the-art Android's recommended toolkit for building native UI. It centralizes the back-end and frond-end code simplifying the management of UI updates based on changes that take place on the back-end side.

Development of Threat Detector follows *MVVM (Model-View-Viewmodel)*[14] which is a modern architectural pattern in computer software that separates the development of a graphical user interface (view) and the business or back-end login (model). According to *Android's developer website*[15], by using instance state mechanism provided by view model, the data is not destroyed upon configuration changes such as navigation between various activities. This ensures data persistent and addresses the configuration change issues that utilizing plain-class-approach introduces.

Testing was performed on a physical device (Samsung Galaxy S22 - Android 15) using the standard APIs available through Android Studio and the following external/third-party APIs:

- **OKhttp[16] & Retrofit[17]** for HTTP communication with AbuseIPDB.
- **GSON[18]** for JSON parsing.
- **Material-Component[19]** for user interface design.

Threat Detector targets Android 12 (API 31) and above to ensure compatibility with the majority of Android devices in use and alignment with new Android's permission and network models.

5.4 PCAPdroid Configuration and Data Export

In this setup PCAPdroid is configured to capture all the network traffic using its internal Android's VpnService implementation. This application supports multiple data export options including file-based (e.g. PCAP/PCAPNG)and network-based exports. For this approach, the TCP exporter and PCAP file format were selected.

The TCP exporter feature provided by PCAPdroid continuously streams packets in real time over a TCP connection socket.

To activate the above-mentioned features, the user should specify:

- **Export protocol**: TCP
- **Host**: 127.0.0.1 (local TCP Server)
- **Port**: 1234 (default port used in PCAPdroid's setting)
- **Format**: PCAP (PCAPdroid extensions should be activated in the captures section)
- **Interval**: None (Continues reception)

After implementing the adjustments mentioned above, when PCAPdroid is launched and activated, it automatically connects to the TCP server established by Threat Detector and continuously transmits captured data as soon as a session begins.

[14] https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel
[15] https://developer.android.com/topic/libraries/architecture/viewmodel
[16] https://github.com/square/okhttp
[17] https://github.com/square/retrofit
[18] https://github.com/google/gson
[19] https://m3.material.io/components

5.5 TCP Server Implementation in Threat Detector

One of the core functionalities of Threat Detector is its TCP server running locally on the Android device. It listens on the incoming connections on a specified port (default set to 1234) and handles each connection request in a separate thread and an associated coroutine ensuring reliable responsiveness and performance.

Simplified version of the server implementation can be viewed in figure Fig. 5.2

```
class TcpServer(private val port: Int) {
    private val serverSocket = ServerSocket(port)

    fun startServer() {
        Thread {
            while (true) {
                val client = serverSocket.accept()
                GlobalScope.launch(Dispatchers.IO){
                    handleClient(serverSocket)
                }
            }
        }.start()
    }

    private fun handleClient(sock: Socket) {
        val input = BufferedInputStream(socket.getInputStream())
        val inputStream = sock.getInputStream()
        val data = inputStream.read()

        Log.d("Retrieved Data", "Data is: $data")
    }
}

```

Figure 5.2: Threat Detector TCP Server Pseudo Code, Own Creation

As shown in the figure above, when a connection is established, a coroutine (`GlobalScope` in this example) starts the continues reception of data in PCAP file format. This is implemented using the `handleClient()` method to extract the low-level information such as destination IP address, timestamps, ports, protocols, and UIDs.

5.6 Parsing of Packet Data

Even thought the packets arrive in PCAP format, when the PCAP extensions is activated, the received data should be reconstructed. This can be carried out with the help of a custom parser developed based on the *documentations*[20] provided for third-party usage by locating

[20] `https://emanuele-f.github.io/PCAPdroid/advanced_features#45-pcapdroid-extensions`

the trailer magic number. After the reception of a packet, the parser reconstructs PCAP packet information into a readable format for ease of data extraction.

Fig. 5.3 shows extracted information including destination IP address alongside HEX-encoded values such as destination port number, source IP address, and the IP payload being the application UID.

```
12:35:23.446  D  IPv4 Destination IP: 20.215.74.200
12:35:23.446  W  in getIpData
12:35:23.447  D  IP: 20.215.74.200
12:35:23.447  D  Score: 0
12:35:23.447  D  Domain: microsoft.com
12:35:23.447  D  Total reports: 0
12:35:23.447  D  Country Code: PL
12:35:23.447  D  Full Packet Dump:
                 0000: 00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00   ..............E.
                 0010: 00 28 00 00 40 00 40 06 23 59 14 D7 4A C8 0A D7   .(..@.@.#Y..J...
                 0020: AD 01 01 BB C2 A2 77 EB A6 BC 35 61 AF DE 50 10   ......w...5a..P.
                 0030: 0F FF C0 18 00 00 00 00 01 07 20 21 00 00 27 98   .......... !..'.
                 0040: 4C 69 6E 6B 20 74 6F 20 57 69 6E 64 6F 77 73 00   Link to Windows.
                 0050: 00 00 00 00 CF 21 61 09                           .....!a.
12:35:23.447  D  Trailer found at index: 56
12:35:23.447  D  Packet size: 88
12:35:23.447  D  UID: 10136
12:35:23.447  D  Retrieved App Name: Link to Windows
```

Figure 5.3: PCAP File Packet Dump, Own Creation

```kotlin
fun isFaangIp(ip: String): Boolean {
    val faangCidrs = listOf(
        // Facebook
        "31.13.24.0/21", "66.220.144.0/20", "69.63.176.0/20", "69.171.224.0/19",
        "74.119.76.0/22", "103.4.96.0/22", "129.134.0.0/16", "157.240.0.0/16",
        "173.252.64.0/18", "179.60.192.0/22", "185.60.216.0/22",

        // Apple
        "17.0.0.0/8",

        // Amazon
        "3.0.0.0/8", "13.52.0.0/16", "13.224.0.0/14", "18.0.0.0/8",
        "52.0.0.0/11", "54.0.0.0/10", "205.251.192.0/19",

        // AbuseIPDB
        "104.26.0.0/16", "172.67.0.0/16",

        // Netflix
        "52.88.0.0/15", "52.26.0.0/16", "34.210.0.0/15", "35.160.0.0/13",

        // Google
        "8.8.8.0/24", "8.34.208.0/20", "8.35.192.0/20", "23.236.48.0/20",
        "34.64.0.0/10", "35.192.0.0/12", "66.102.0.0/20", "72.14.192.0/18",
        "74.125.0.0/16", "108.177.8.0/21", "172.217.0.0/16", "173.194.0.0/16",
        "192.178.0.0/15", "199.36.154.0/23", "216.58.192.0/19"
    )

    return faangCidrs.any { cidr -> isIpInRange(ip, cidr) }
}//isFaangIp
```

Figure 5.4: Method To Check IPs Against FAANG Addresses, Own Creation

In Fig. 5.3 the extracted information including the destination IP addresses are added to a queue for further analysis and queries from AbuseIPDB.

Worth to mention that the IP addresses are cached to prevent redundant API calls and they are filtered against FAANG (Facebook-Meta, Apple, Amazon, Netflix, Google) IP addresses that are frequently utilised by third-party applications to save API calls. This is illustrated in Fig. 5.4.

5.7 Integration With AbuseIPDB API

AbuseIPDB[21] service provides a RESTful API to check the maliciousness of the IP addresses based on their public reports. Threat Detector uses this API to determine whether an IP address is potentially malicious.

The way Threat Detector queries the IP addresses is as follows:

```
GET https://api.abuseipdb.com/api/v2/check?ipAddress={IP}&maxAgeInDays=90

Headers: "Key"
Key: <API key>
Accept: application/json
```

Fig. 5.5 shows how the HTTP query is built in Threat Detector.

```
val client = OkHttpClient.Builder()
    .addInterceptor { chain ->
        val request = chain.request().newBuilder()
            .addHeader( name = "Key",  value = apiKey)
            .addHeader( name = "Accept",  value = "application/json")
            .build()
        chain.proceed(request)
    }
    .build()
```

Figure 5.5: Querying IP Address using HTTP GET Method, Own Creation

AbuseIPDB responses vary based on the type of request sent to the server. Different types of requests can be found in the official *AbuseIPDB documentation*[22].

The request used by Threat Detector is "CHECK endpoint" which yields the following JSON response as shown in Fig. 5.6.

[21] https://www.abuseipdb.com/
[22] https://docs.abuseipdb.com/?shell#introduction

```
{
  "data": {
    "ipAddress": "118.25.6.39",
    "isPublic": true,
    "ipVersion": 4,
    "isWhitelisted": false,
    "abuseConfidenceScore": 100,
    "countryCode": "CN",
    "countryName": "China",
    "usageType": "Data Center/Web Hosting/Transit",
    "isp": "Tencent Cloud Computing (Beijing) Co. Ltd",
    "domain": "tencent.com",
    "hostnames": [],
    "isTor": false,
    "totalReports": 1,
    "numDistinctUsers": 1,
    "lastReportedAt": "2018-12-20T20:55:14+00:00",
    "reports": [
      {
        "reportedAt": "2018-12-20T20:55:14+00:00",
        "comment": "Dec 20 20:55:14 srv206 sshd[13937]: Invalid user oracle from 118.25.6.39",
        "categories": [
          18,
          22
        ],
        "reporterId": 1,
        "reporterCountryCode": "US",
        "reporterCountryName": "United States"
      }
    ]
  }
}
```


Figure 5.6: An Example of AbuseIPDB JSON Response [6]

The reputation score, marked as `abuseConfidenceScore`, is then compared with a user-defined threshold stored in `SharedPreferences`. If the score exceeds the defined threshold, it is flagged as potentially malicious and is then followed by a notification process to inform the user.

5.8 Data Storage and Application Configuration

Threat Detector uses Android's `SharedPreferences`[23] to store user's preferences on the device.

These Preferences are as follows:

- TCP server port
- Minimum Confidence Score that user considers to be malicious
- AbuseIPDB API key

[23] `https://developer.android.com/training/data-storage/shared-preferences`

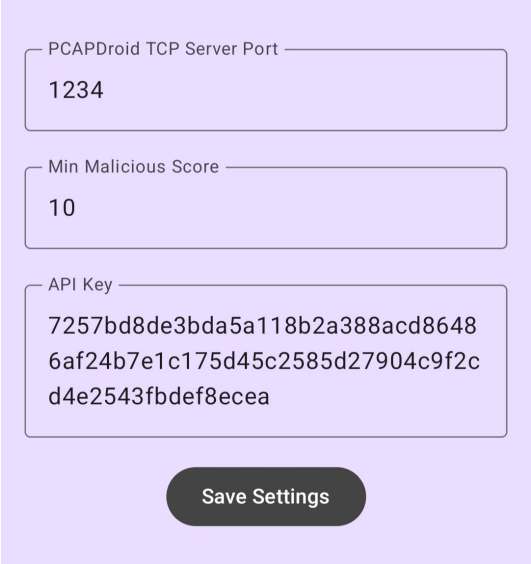


Figure 5.7: Threat Detector Saved Preferences, Own Creation

Android's `SharedPreferences` utilises a dictionary approach to store reusable and modifiable data onto the Android device. It requires a *Key* and an associated *Value* that will be stored in the application data folder. The use of `SharePreeferences` is feasible when there is a small collection of information that does not justify the employment of a local database such as *Room*[24] which simplifies the implementation of an SQL database to store application-related information.

Fig. 5.8 shows the use of `SharedPreferences` in Threat Detector.

```
// Save to SharedPreferences
val sharedPref = context.getSharedPreferences( name = "AppSettings", mode = Context.MODE_PRIVATE)
with( receiver = sharedPref.edit()) {
    putInt("tcpServerPort", tcpServerPort)
    putInt("abuseIpDbMaliciousScore", abuseIpDbMaliciousScore)
    apply()
}
```

Figure 5.8: Threat Detector SharedPreferences Kotlin Code, Own Creation

This approach of storing settings and preferences avoids the complexity of establishment and management of a local database, for the application is designed to be lightweight.

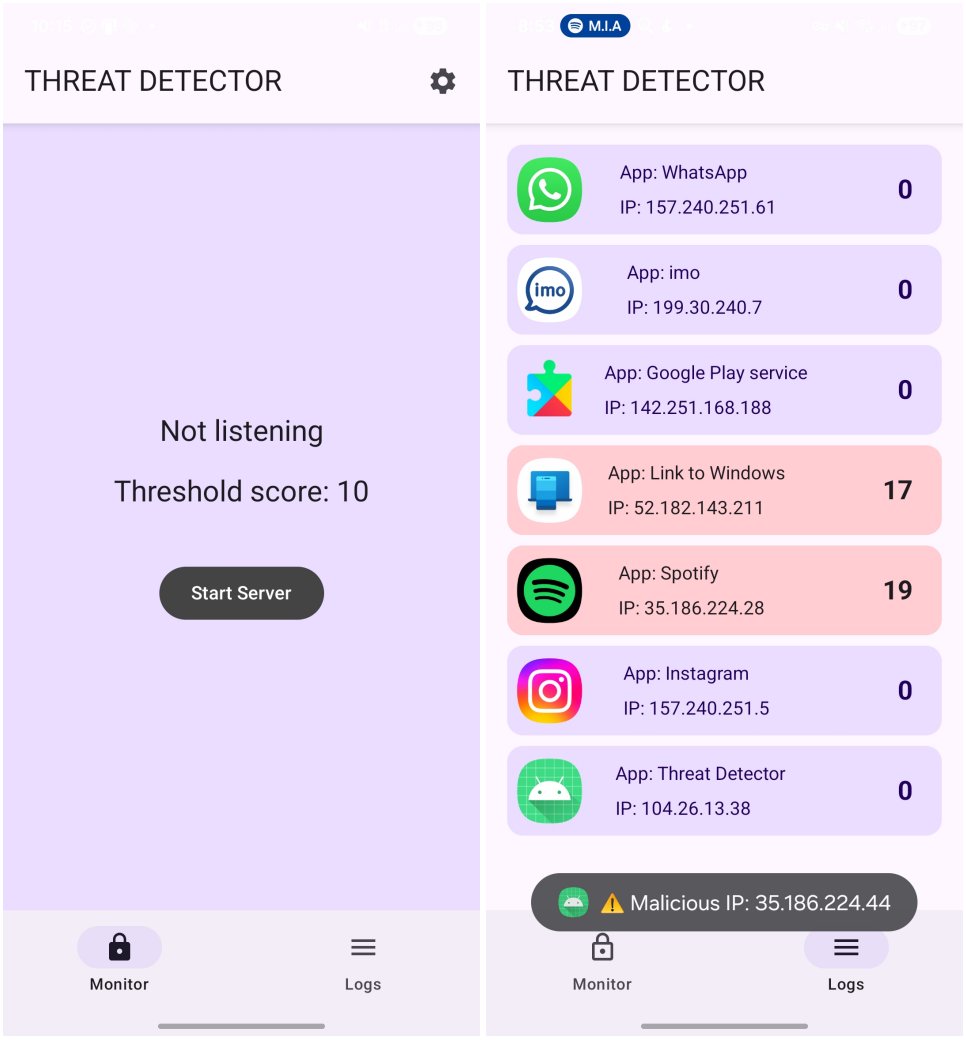
5.9 User Interface Implementation

Threat Detector's user interface was designed based on *Material Design 3*[25]. It is Google's open-source design system for building UI components with vibrant colours, contrasting shapes, flexible typography, and intuitive motion.

[24] `https://developer.android.com/training/data-storage/room/`
[25] `https://m3.material.io/`

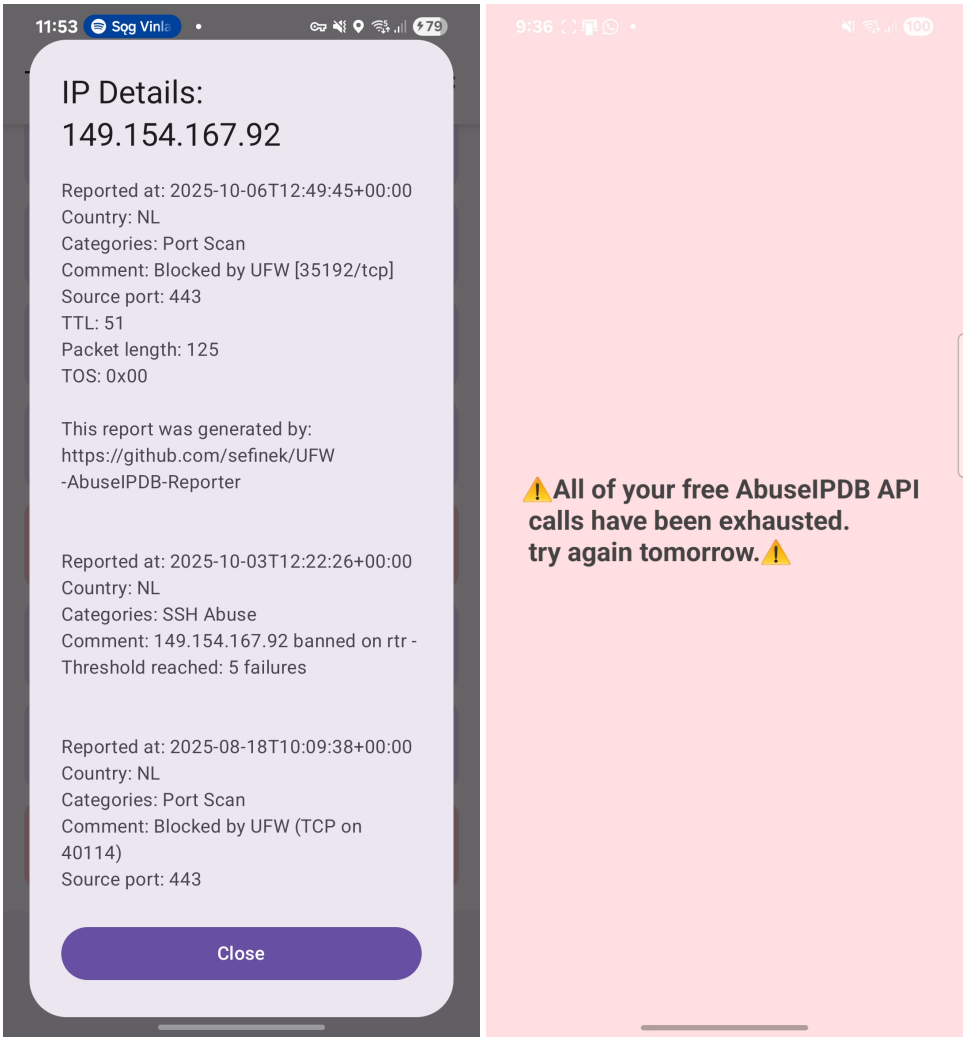
Threat Detector's interface consists of the following sections:

- **Monitor Page - Fig. 5.9a:** provides the functionality to start/stop the local TCP server alongside depicting the server activation status and user-defined IP confidence score.
- **Logs Page - Fig. 5.9b:** provides user with a scrollable list of the applications that attempted to communicate with external IP addresses. Each list Entry shows the application name and icon along with the IP address and its confidence score.
- **IP Details Page - Fig. 5.10a:** depicts additional information regarding the queried IP address including the countries that the IP address has been reported in, categories of attack, source port, etc.
- **API Exhaustion Page - Fig. 5.10b:** it appears as a notification page when the user has exhausted the free daily API calls from AbuseIPDB.
- **Settings Page - Fig. 5.8:** it is used to save user's preferences such as TCP server port, API key, and minimum IP confidence score.



(a) Monitor Page (b) Logs Page

Figure 5.9: Screenshots from Threat Detector, Own Creation



(a) IP Details Page (b) IP Exhaustion Page

Figure 5.10: Screenshots from Threat Detector, Own Creation

The user interface components update dynamically as new incoming data is registered by the TCP server. This includes the real time automatic updates to modify the items shown in Logs page, the colour change indicating the listening status in Monitor page, and the illustration of IP Exhaustion Page as a pop-up when the user runs out of free API calls.

5.10 Communication Flow and Synchronization

The communication between Threat Detector and its companion application, PCAPdroid, takes place in a purely local context. The TCP connection socket is established through a loopback interface provided by Android's basic APIs, which ensures that no external component is involved. The server runs in a background thread, so that managing its lifecycle and configuration changes do not block the main UI thread leading to performance flaws.

In the case that PCAPdroid disconnects, stops capturing, or even stops data transmission, the server automatically cleans up the socket and its associated stream buffers and waits

for new connection requests. Moreover, error handling routines are also taken into consideration for detection of malformed packets and connection timeouts.

5.11 Security and Privacy Consideration

Threat Detector is designed with conservation of user's privacy in mind. All the captured network data is processed locally and nothing is uploaded to an external server, except AbuseIPDB which is strictly IP-based and does not contain any personal identifiers/information.

Some additional precautions include:

- Secure storage of the API key in private application folder.
- Local communication is restricted to local host (127.0.0.1).
- Use of HTTPS for external queries from AbuseIPDB.
- No conservation of raw PCAP data than real-time analysis.

5.12 Performance Optimization

Since Threat Detector operates in real time, efficiency of the operations were of concern.

To ensure an optimized functionality, the following strategies were adopted:

- Asynchronous querying of AbuseIPDB utilising background threads to prevent UI blockage.
- Caching of the recent IP lookups to avoid redundant requests.
- Custom lightweight parsing of PCAP metadata instead of a full reconstruction of the packets.
- Use of `InputStream` and buffers to implement thread-safe queues for data reception.

5.13 Limitations

In spite of the successful implementation of Threat Detector, this application introduces some fundamental limitations that can be of importance for further development of such monitoring and reputation-checking solution.

The limitations are as follows:

- Threat Detector is an extension for PCAPdroid meaning that it is totally dependent on PCAPdroid for packet capture and PCAP file construction; Threat Detector cannot operate independently and requires the installation of PCAPdroid on the same device.
- AbuseIPDB free subscription enforces query rate limits (1000 queries per day), which restricts large-scale monitoring. However, this limitation can be solved by subscribing to paid plans.
- Payload content inspection is not carried out since most of the applications use encryption (e.g. TLS/SSL) between endpoints.

- Use of `SharedPreferences` rather than a dedicated database restricts historical tracking of the modifications carried out on stored preferences.

5.14 Summary

The implementation of Threat Detector and its integration with PCAPdroid provide a practical solution for monitoring and assessing the network traffic on Android devices. Its modular design allows further development and integration with other security tools that make use of TCP connections for data transmission. It successfully leverages AbuseIPDB's reputation intelligence and effectively bridges the gap between low-level packet capture and real-time threat awareness and user notification.

While Threat Detector is a lightweight and privacy-preserving solution that implements threat intelligence, it lays the foundation for several enhancements such as IP blocking, application-specific payload visualization, and AI-based anomaly detection.

Overall, this project illustrates the feasibility of real-time mobile traffic analysis in the form of a hybrid approach using open-source tools and external reputation-checking databases.

6

Evaluation and Discussion

7 Conclusion

In the conclusion, all the main results are summarised once again. Here, experiences made can also be described. At the end of the summary, an outlook can also follow, which presents the future development of the topic dealt with from the author's point of view.

List of Figures

List of Tables

Bibliography

[1] Android Developer Website. Vpnservice vpn, 2025. URL https://developer.Android.com/develop/connectivity/vpn. Accessed: 25 October 2025.

[2] PCAPdroid Website. Pcapdroid user guide, 2025. URL https://emanuele-f.github.io/PCAPdroid/quick_start.html. Accessed: 25 October 2025.

[3] AthinaGroup. Antmonitor website, 2025. URL https://athinagroup.eng.uci.edu/projects/antmonitor/. Accessed: 25 October 2025.

[4] Daoyuan Wu, Weichao Li, Rocky K. C. Chang, and Debin Gao. Mopeye: Monitoring per-app network performance with zero measurement traffic, 2016. URL https://arxiv.org/abs/1610.01282.

[5] Daoyuan Wu, Rocky K. C. Chang, Weichao Li, Eric K. T. Cheng, and Debin Gao. MopEye: Opportunistic monitoring of per-app mobile network performance. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 445–457, Santa Clara, CA, July 2017. USENIX Association. ISBN 978-1-931971-38-6. URL https://www.usenix.org/conference/atc17/technical-sessions/presentation/wu.

[6] AbuseIPDB.