
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: | 2025-10-04 |
| 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

implement the correct citation of the websites

## 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 assistance, 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, both dynamic and 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 they can 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 endanger 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* and *Ant-Monitor* 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. 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 processing 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 ensures 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' VpnService API provided by a parent 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 enhanced 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 enquiry 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 comprehensively 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* 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 into 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 the escalate into 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 ensures a vast target audience reluctant to root their android devices.

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 message and does not have direct access to the aforementioned API, lack of blocking capabilities still persists as an inhibiting 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*. As a result, it is made possible to receive the PCAP/PCAPNG file in the JSON format via a local TCP/UDP servers.

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 the 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 Paper

This paper 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 from the initial idea and the deviations, followed by some modifications, to the realization of the application. It covers how the design metrics and architecture are carried out to satisfy efficient performance, implement user-friendly UI (user interface) and comprehensive coordination with PCAPdroid. It exhibits UI components and elaborates on the back-end functionalities they execute.
- ***Setup with PCAPdroid:*** This section goes through a thorough 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 working to 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 impossble 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 and also represents a foundation for future work, research and actions in securing android devices in our increasingly complex and hyperconnected environments.

# 2 Background

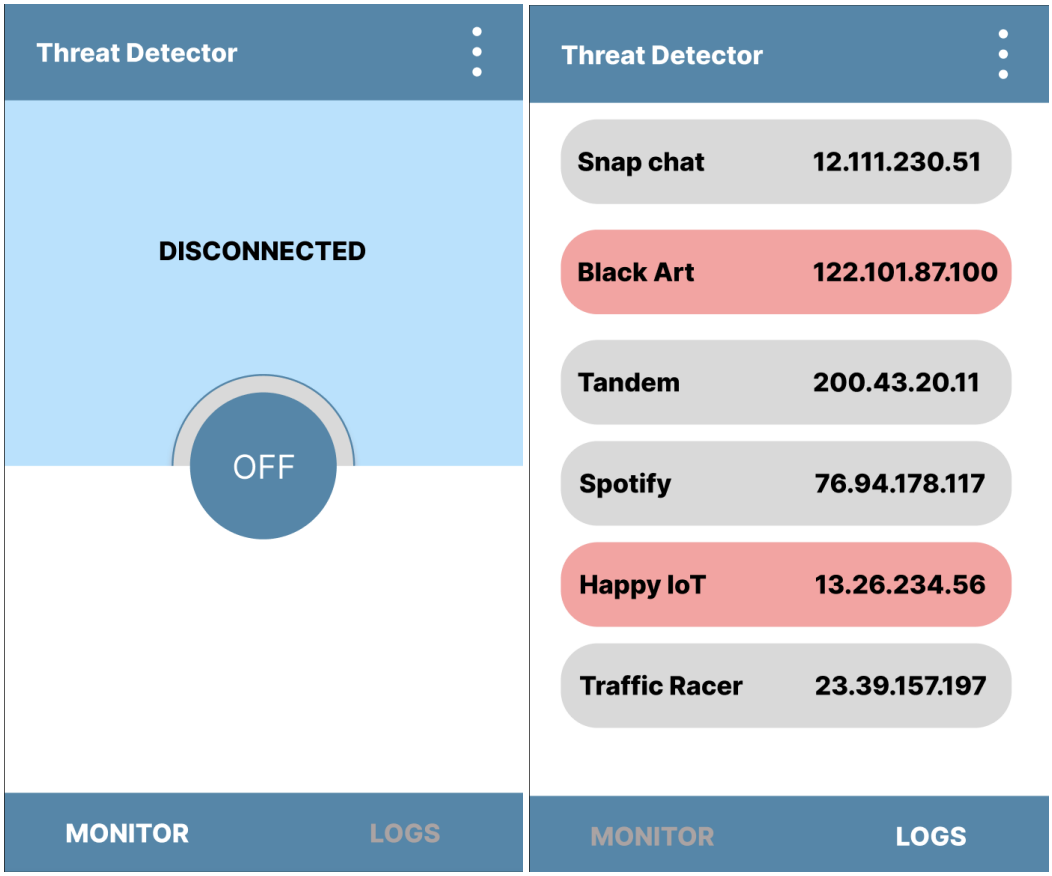
This Chapter provides a background and the motivation for development of Threat Detector. It outlines some of the fundamental gaps in the current mobile network monitoring and packet inspection systems and the unique challenges they faced which led to the creation on this novel approach. This chapter 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 vital aspects such as packet-to-application mapping, real-time threat detection and reputation checking and user-friendly implementation of threat intelligence. 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 application activity.

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




(a) Monitor Page (b) Logs Page

Figure 2.1: Threat Detector UI based on initial idea

### 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 documentation*, 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.

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

```
// 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

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)
```

(a) Defining Default Gateway

```
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()
        }
    }
```

(b) Reading Incoming/Outgoing Data

Figure 2.3: Usage of VpnService API

```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 2.4: Extracting Destination IP Address from Raw Packets

In the screenshots provided in Figure 2.3, 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.

Figure 2.4 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 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 of 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 determining a default gateway, routing all the traffic through it, transmitting packets across network using IP packet encapsulation, using 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 deemed 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.

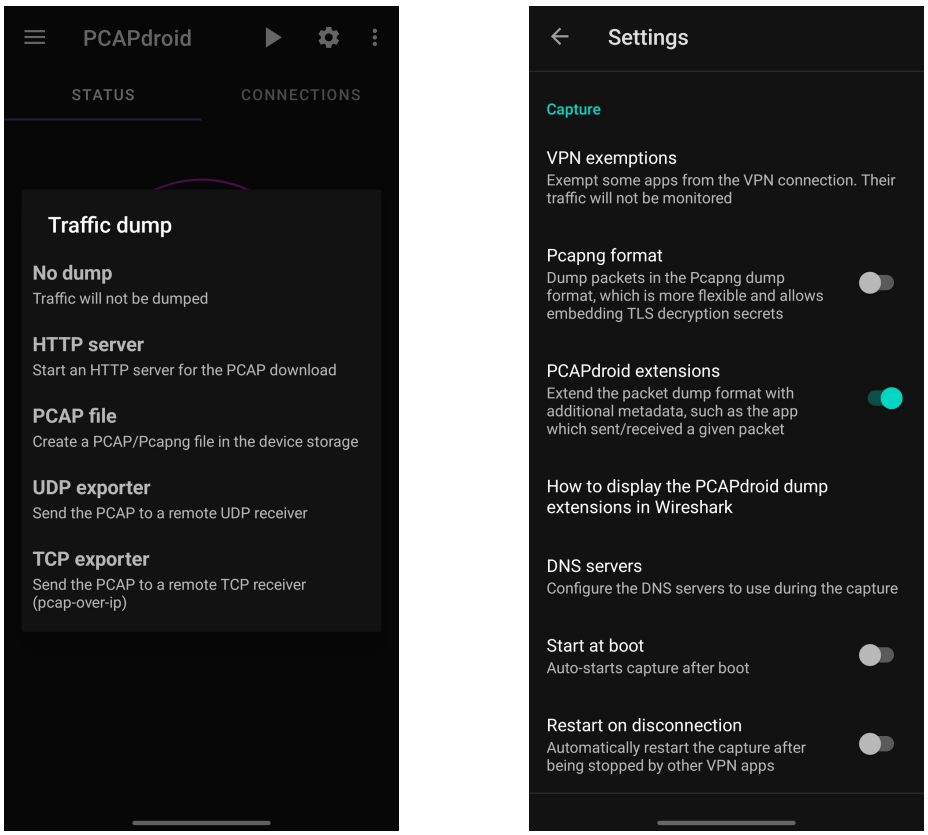
From the above-mentioned options, none of them is feasible to use. Rooting the android device for a normal user brings along a lot of drawbacks including weakened security and extended susceptibility to various attack surfaces. Moreover, as of android version 11 and above, the system-level permissions 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* 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, Mobeye, 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.




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

Figure 2.5: Usage of VpnService API

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, among other 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, Mobeye***, 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, Mobeye aims its focus towards large-scale network measurements and data aggregation for further research and monitoring purposes. It contributes valuably in 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 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 be 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.

## 3.2 Mobeye

## 3.3 AntMonitor

# 4

# Architecture/System Design

# 5 Implementation

# 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.

# A

# Appendix

# List of Figures

# List of Tables

# Bibliography

[1] Benjamin Taubmann, Noelle Rakotondravony, and Hans P. Reiser. CloudPhylactor: harnessing mandatory access control for virtual machine introspection in cloud data centers. In *2016 IEEE Trustcom/BigDataSE/ISPA*, pages 957–964, Aug 2025. doi: 10.1109/TrustCom.2016.0162.