Project Report

Development of an Android Application for Detecting PII Leaks

Project Start Date	Project End Date
February 20, 2024	July 20, 2024

Prepared By
Bhartendra Sharma
Nikhil Mangal
Lakshya Sharma
Prachi Rathore

Supervised By
Dr. Jyoti Gajrani
HOD CSE
GEC Ajmer

Prepared For
Dr. Manoj Singh Gaur
Director
IIT Jammu

Introduction

Our team was honoured to be invited to work on the development of an Android application for detecting PII leaks. This project, initiated on February 20, 2024, has been a collaborative effort involving continuous guidance and support from esteemed faculty and experts.

We were brought on board for this project by Dr. Jyoti Gajrani, whose able guidance and mentorship have been instrumental in our research and development activities. Throughout the course of this project, we have had the privilege of holding regular and insightful meetings with Dr. Shaifu Gupta and Ms. Rishika Kohli. These meetings have provided us with valuable feedback and direction, enabling us to make significant progress.

Under the expert guidance of Dr. Jyoti Gajrani and with the continuous support from Dr. Shaifu Gupta and Ms. Rishika Kohli, our team has been dedicated to developing a robust solution for detecting PII leaks in phone applications. This report details our progress, methodologies, and the outcomes of our efforts thus far.

Progress Overview

First Meeting (February 20, 2024)

- Our first meeting with the Dr. Shaifu Gupta and Ms. Rishika Kohli took place on February 20, 2024. During this meeting, we were provided with a comprehensive overview of their existing research and the objectives they aimed to achieve through our collaboration.
- We were introduced to two essential tools, **AntMonitor** and **AntShield**, which are installed on user phones to monitor all applications on the device. These tools detect any PII leaks and inform the user about the specific app and the personal information that has been leaked.
- A sample JSON file from the antshield was also provided as we were asked to deliver the captured packet in the same format and order.
- The requirements and expectations were clearly defined, setting the stage for our subsequent research and development activities.

Antmonitor

AntMonitor is a user-space mobile application designed for real-time monitoring and analysis of network traffic to detect and prevent private information (PII) leaks from mobile devices. It operates without requiring root access by utilizing a VPN service to capture and analyse all network packets on the device. This approach ensures ease of installation and use while maintaining high performance and scalability.

Antmonitor Testing Model

Device and Android Version

- The tests were conducted on a Nexus 6 with Android 5.0. Since this is an older device and Android version, the results may not be representative of newer devices or Android versions.

Controlled Environment

- The experiments were performed in a controlled environment (late-night hours, no Google sign-in, minimal apps). This might not reflect real-world usage scenarios where background traffic and diverse app usage can impact performance.

TLS Interception Disabled

- TLS interception was disabled, which means the tests did not account for the inspection of encrypted traffic. This could be a significant limitation, as encrypted traffic is common in real-world applications.

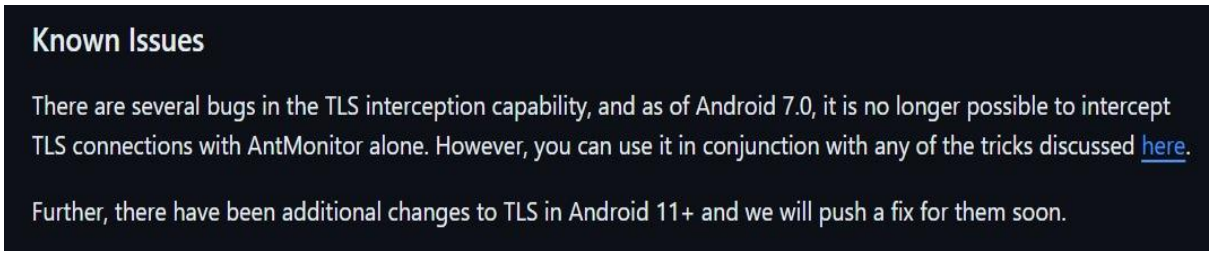
Compatibility Issues

- AntMonitor faces significant compatibility issues with newer Android versions, starting from Android 7.0 and becoming more pronounced with Android 11+.
- These issues primarily revolve around the inability to intercept TLS connections, which are critical for monitoring encrypted traffic.

Setup. All experiments were performed on a Nexus 6, with Android 5.0, a Quad-Core 2.7 Ghz CPU, 3 GB RAM, and 3220 mAh battery. Nexus 6 has a built-in hardware sensor, Maxim MAX17050, that allows us to measure battery consumption accurately. Throughout the experiments, the device was unplugged from power, the screen remained on, and the battery was above 30%. To minimize background traffic, we performed all experiments during late night hours in our lab to avoid interference, we did not sign into Google on the device, and we kept only pre-installed apps and the apps being tested. Unless stated otherwise, the apps being tested had TLS interception disabled and the AntMonitor App was logging full packets of all applications and inspecting all outgoing packets. In terms of versions, all tests were done with AntMonitor v0.0.1 and Haystack v1.0.0.8, unless indicated otherwise (Sec. IV-A1). VPN servers ran on a Linux machine with 48-Core 800 Mhz CPU, 512 GB RAM, 1 Gbit Internet; the Wi-Fi network was 2.4Ghz 802.11ac. Each test case was repeated 10 times and we report the average.

Conclusions

While AntMonitor was a pioneering tool for detecting PII leaks on mobile devices through a VPN-based approach, the advancements in Android's security and privacy measures have made it increasingly difficult for such tools to function as intended. The limitations on VPN services, enhanced encryption methods, and stringent resource management policies have collectively contributed to the reduced applicability of AntMonitor on current Android devices.



~Anitmonitor Git Repository

Antshield

We present AntShield - a system that performs efficient on-device analysis, provides accurate and comprehensive data privacy protection, and gives users transparency and control over their personal information in real-time. A key insight is the distinction between PII that is predefined by the user or is readily available on the device, from PII that is a priori unknown and should be inferred by classifiers. We propose a hybrid String Matching-classification approach we build on the AntMonitor Library for intercepting packets on the device and looking for predefined strings in real-time and (ii) we build classifiers for the remaining unknown PII.

In simple words the Antshield is a framework which is using Antmonitor as a library for capturing and intercepting network traffic and after capturing all of the data it is using recon a machine learning based tool who has already been trained on the dataset of PII which perform hybrid DPI (DEEP PACKET INSPECTION) using predefined classifiers.

We were assigned with the task to provide the Json in the same format as the antshield's json but the code of antshield was not public. So, we started to search for the alternatives of Antshield.

1. Powertunnel
2. PcapDroid

Powertunnel

PowerTunnel was the next tool we tested for our project, aiming to detect and log PII leaks from phone applications. While the application ran successfully on our mobile devices, we encountered significant challenges in capturing and storing logs.

Issues Encountered

1. **Log Capture Failure**
    - Although the application was running smoothly, we could not capture the logs directly within the PowerTunnel app.
    - Logs were only visible in the logcat of Android Studio during the app's execution, but there was no mechanism to store these logs within the app itself.
2. **PCAP File Storage Issue**
    - Due to the log capture failure, we were unable to generate and store logs in a PCAP (Packet Capture) file.
    - The absence of PCAP files meant we couldn't analyse the payloads for encryption and decryption statuses, which is crucial for assessing the security and privacy of data transmission.

Limitations Identified

- **No Direct Log Storage** The inability to capture and store logs within the app limited our ability to perform offline analysis and verify the extent of encrypted and decrypted data.
- **Dependency on Logcat** Relying on Android Studio's logcat for log visibility is not practical for real-world deployment and continuous monitoring, as it requires a connected development environment.

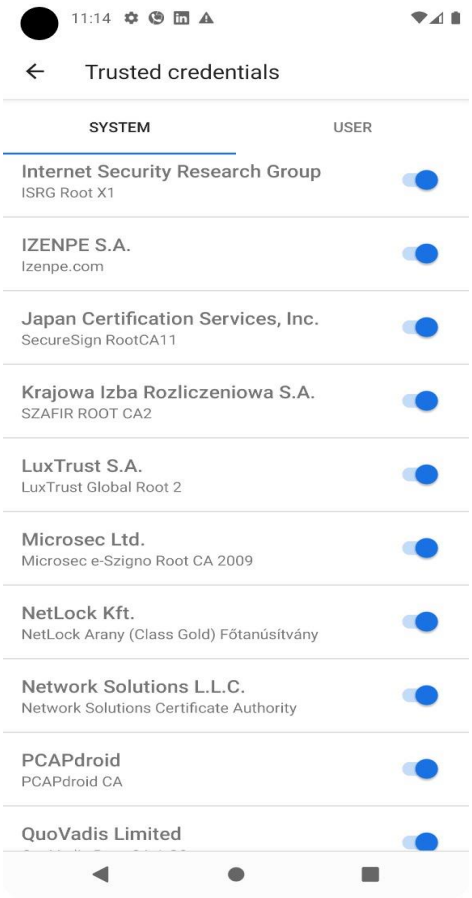
Conclusion While PowerTunnel shows potential as a monitoring tool, its current limitations in log capture and storage hinder its effectiveness for our project's requirements. Future iterations or alternative tools need to address these issues to provide a comprehensive solution for detecting PII leaks and analysing data security.

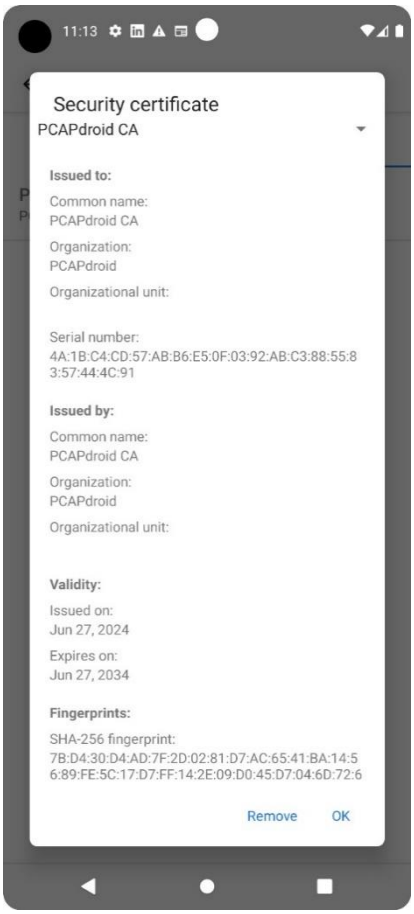
---

*PCAPDROID

As we explored alternatives to AntShield, we discovered PcapDroid, which offers a local VPN service like that of AntShield. PcapDroid utilizes a MITM proxy for TLS decryption, while AntShield employs Sandro Proxy for this purpose. During our testing of PcapDroid on both system and third-party apps, we found that most of the traffic was TLS encrypted. Consequently, we were unable to decrypt the data to determine if any PII was present.

After extensive testing on multiple applications, we failed in decrypting any traffic because the applications did not trust the MITM certificate, which was initially installed in the user-installed certificates section. According to the new rules and regulations, it has become mandatory for Android applications above Android 7 to not trust certificates outside the system-installed certificates. Therefore, to achieve decryption, it has become necessary for us to perform rooting on the devices.





Testing Environment

1. Android Studio Emulator
2. Device Google Pixel 7
3. Android 11
4. API Level 30

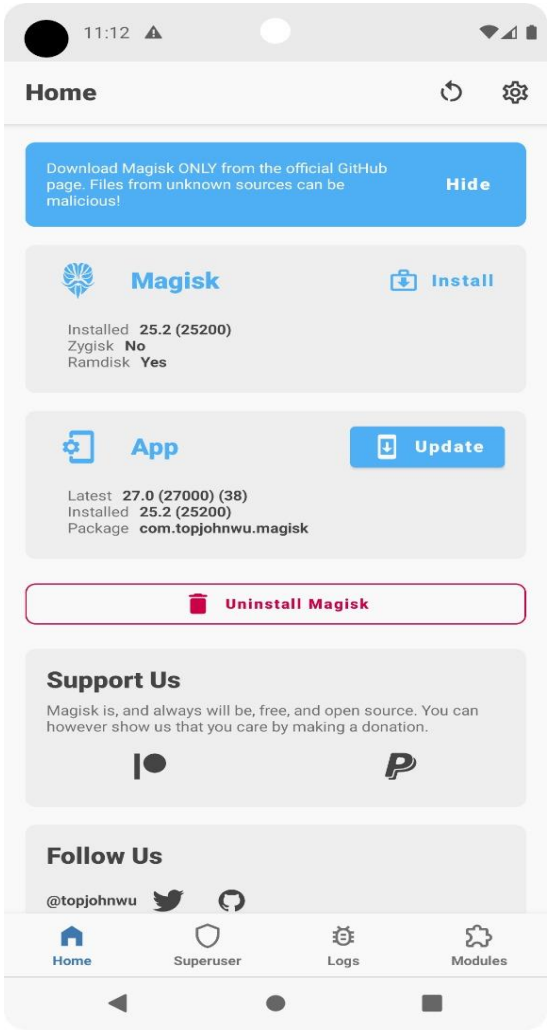
Rooting Process Root AVD (Magisk Rooting App)

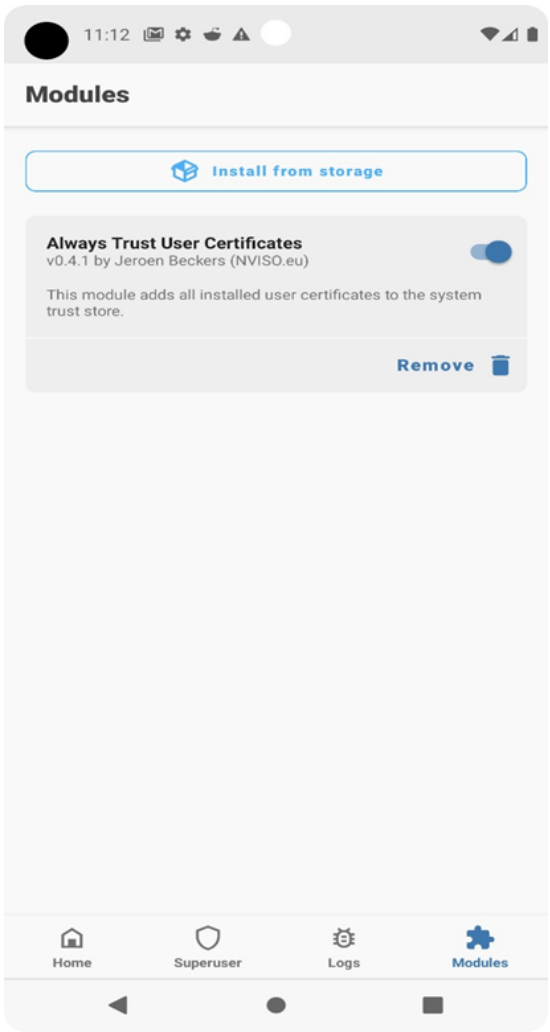
Detailed Procedure for Traffic Analysis and JSON Conversion Using PcapDroid:

1. **Rooting the Device and Installing Magisk**

- Root the device and install the Magisk app.

- Activate "Trust User Certificates" within Magisk to allow user certificates installation in the system.





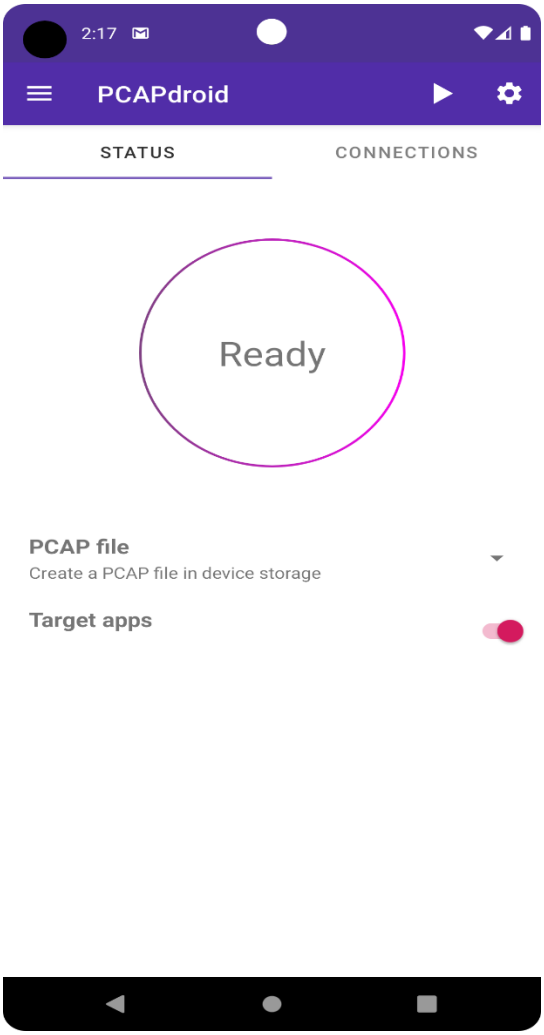
2. **Installing PcapDroid**

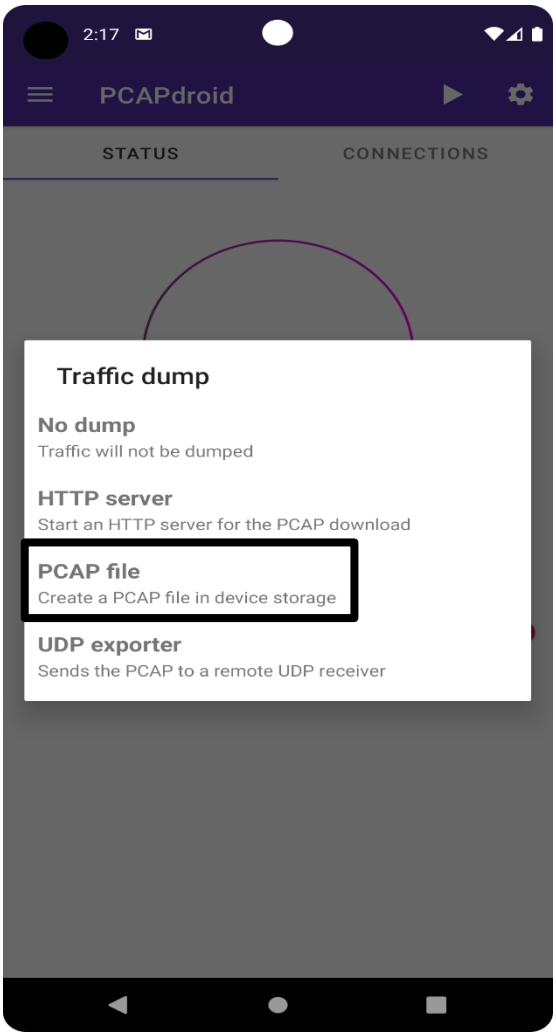
- Install PcapDroid on the emulator.

- On the PcapDroid homepage, configure

  - Target Apps Select specific applications to monitor.

  - PCAP File Location Choose where to store the captured traffic.

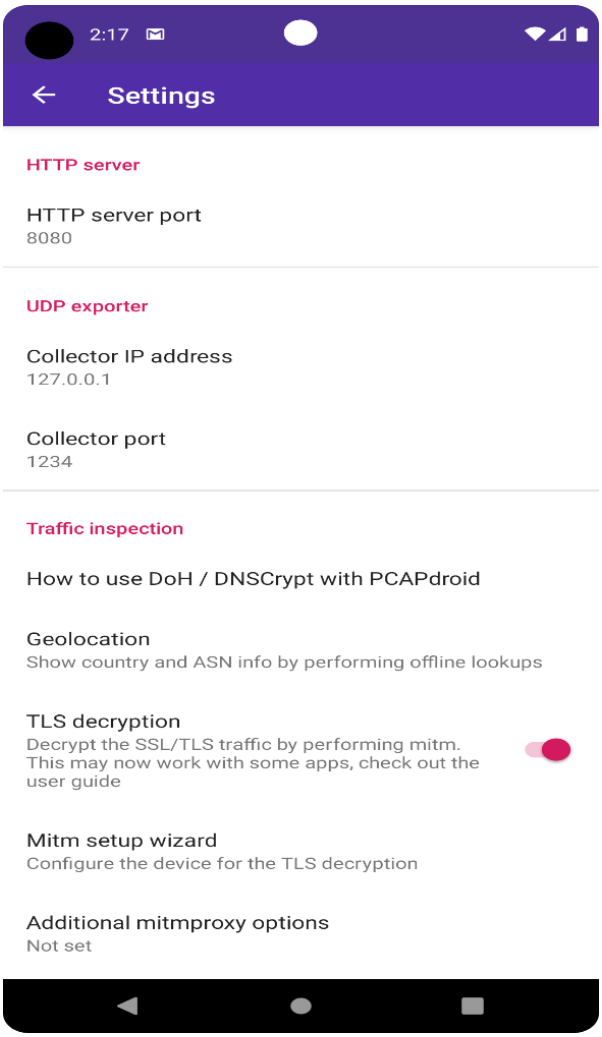




3. **Configuring PcapDroid**

- In the settings menu, locate the TLS decryption toggle switch and turn it on (default is off).

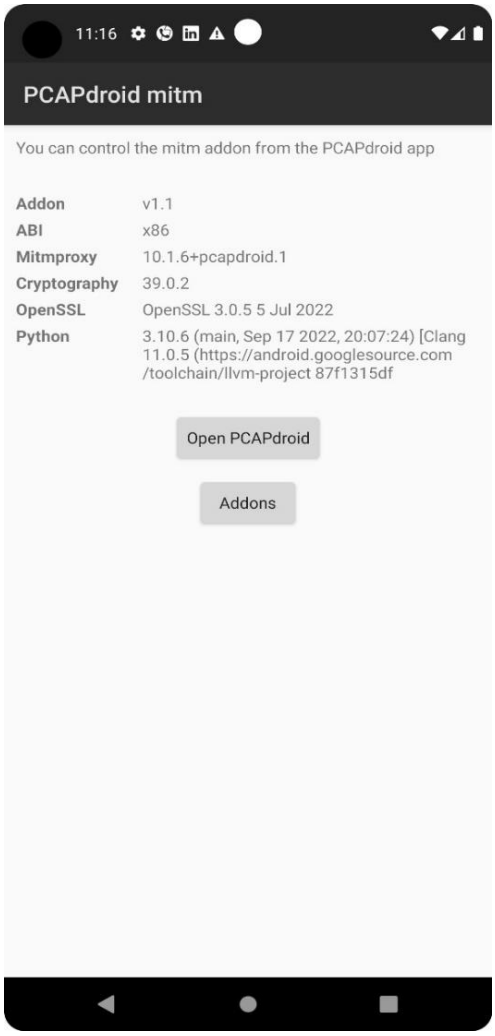
- Follow PcapDroid's guide to set up a MITM proxy for TLS decryption.



4. **Setting Up MITM Proxy**

- PcapDroid guides the setup of the MITM proxy application and installation of MITM trust certificates in the user space.

- The Magisk module transfers these certificates to system-installed certificates.



5. **Capturing Traffic**

- Enable settings

  - Capture as Root

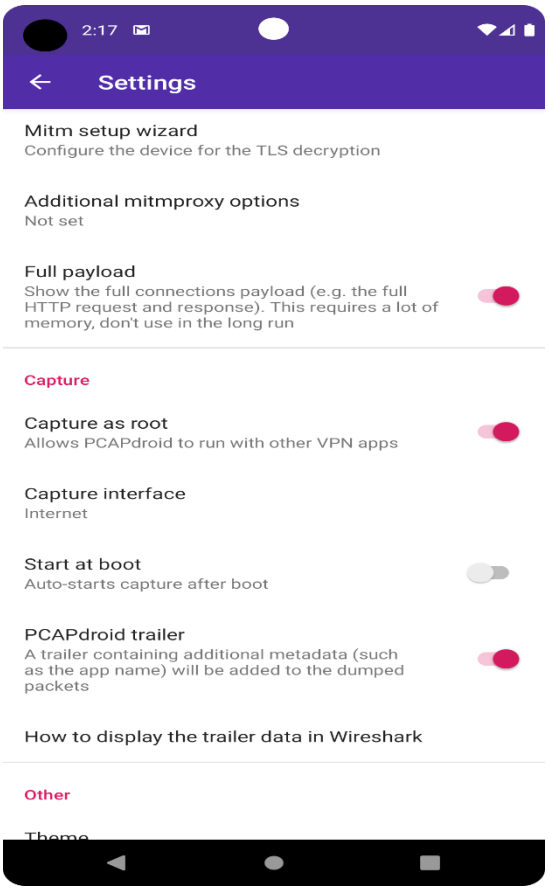
  - Capture Full Payload

- Click the start button to begin capturing traffic.

- Use the target application to generate traffic.

- Stop the capture after sufficient data is collected.

- Save the traffic in a PCAP file along with an `ssllogkey.txt` file.



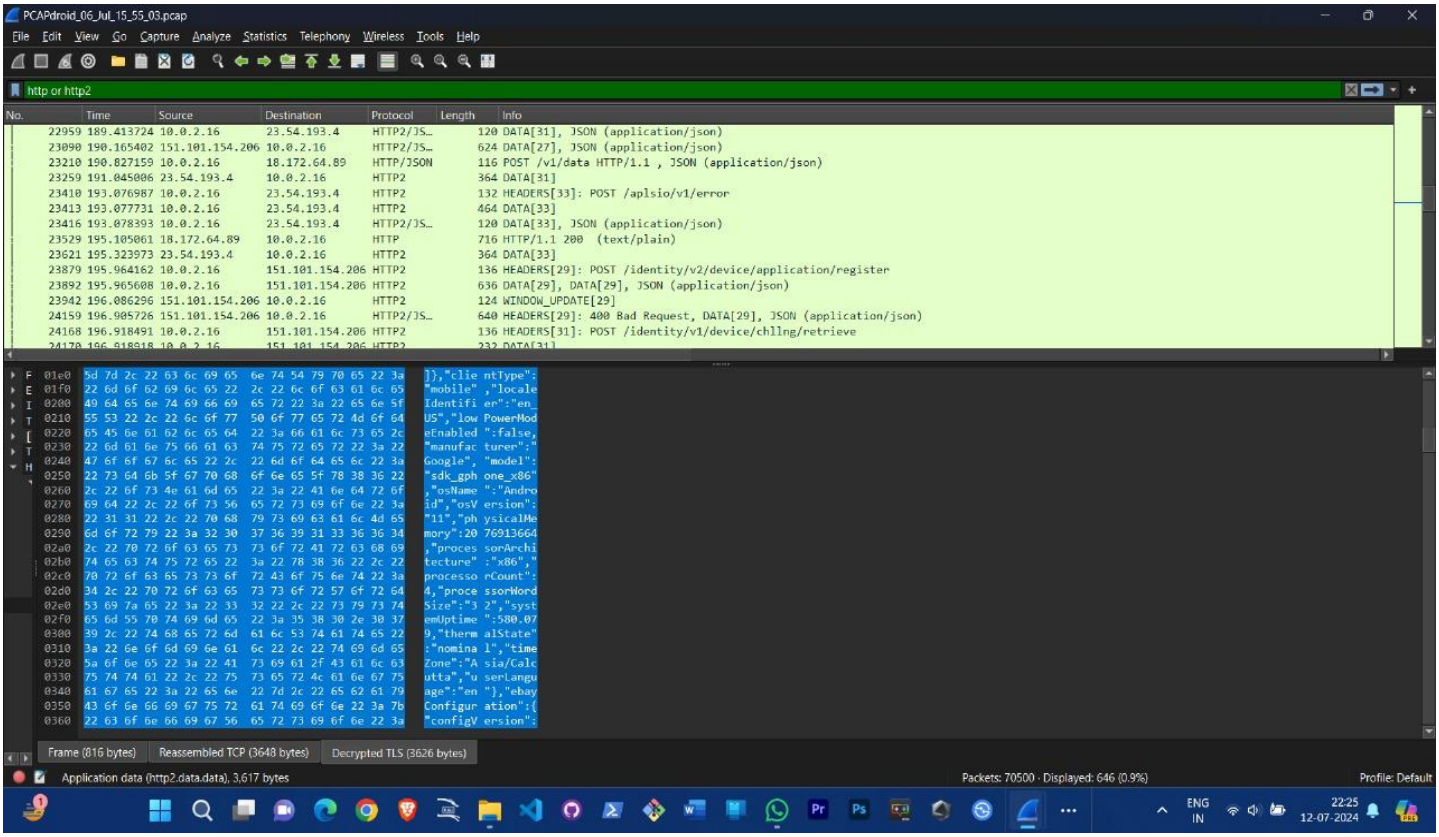
6. **Analysing Captured Traffic**

- Load the PCAP file into Wireshark.

- Add the `ssllogkey.txt` file to Wireshark to decrypt TLS traffic.

- Decrypted TLS traffic appears as HTTP traffic which holds the plain text.

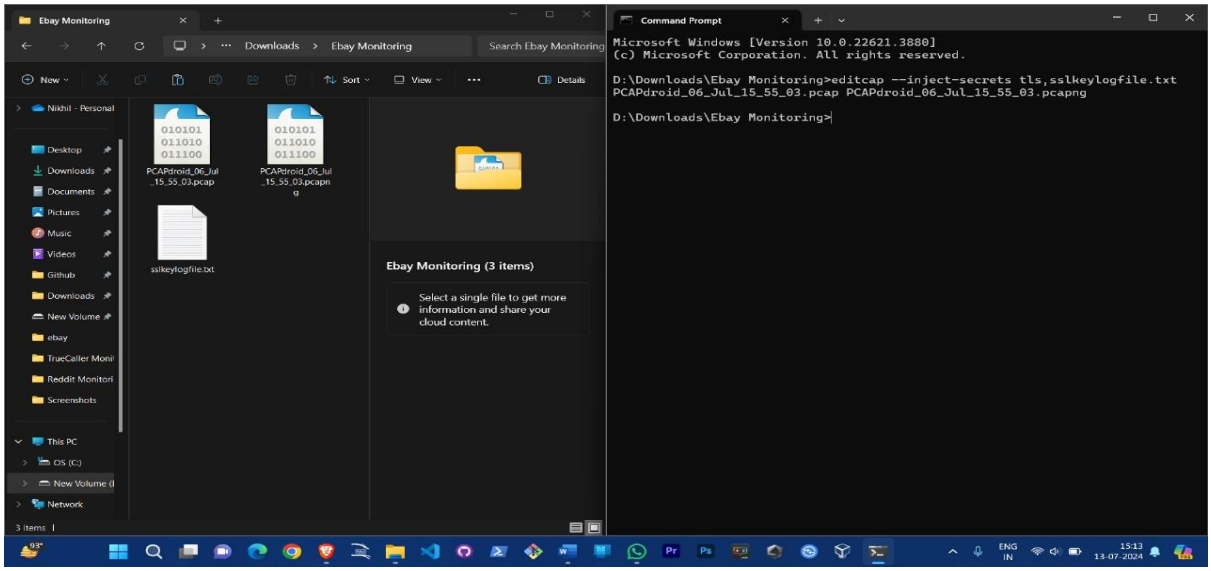
- Manually inspect the decrypted data for PII.



7. **Converting PCAP to PCAPNG**

- Convert the PCAP file to a PCAPNG file, which combines the PCAP file and the SSL log key for enhanced analysis.

```
editcap --inject-secrets tls, keys.txt in.pcap out-dsb.pcapng
```

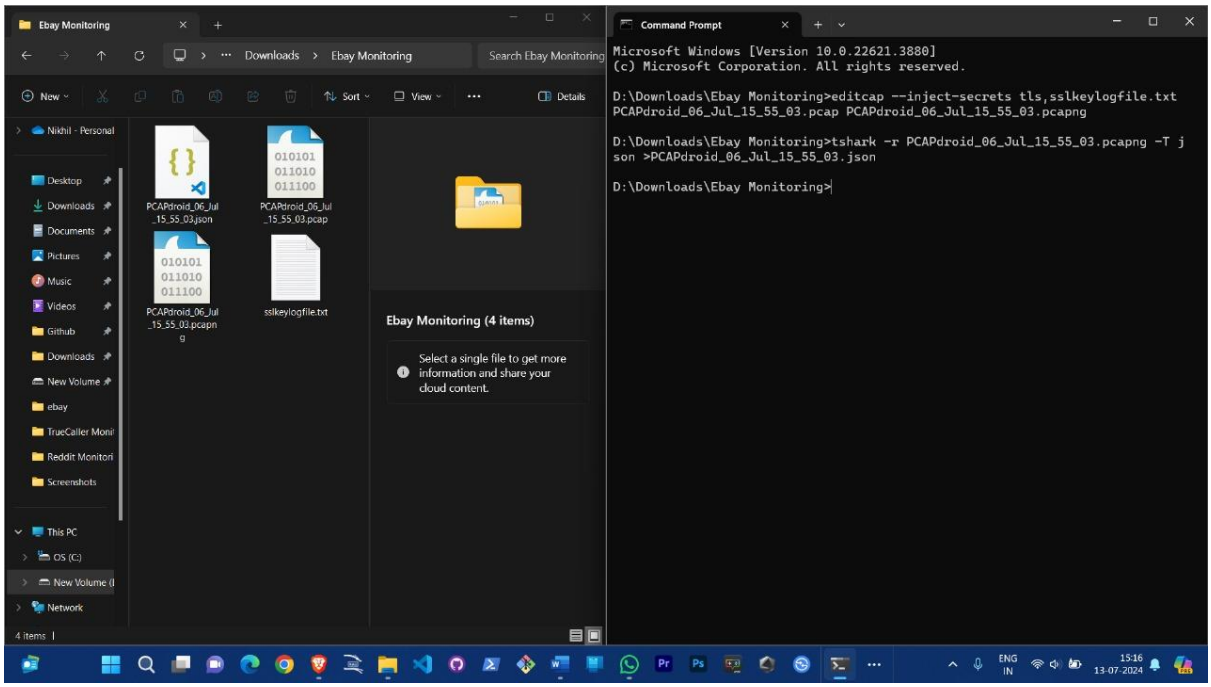


8. **Converting PCAPNG to JSON**

- Use Tshark, the command-line version of Wireshark, to convert the PCAPNG file to a JSON file

```
tshark -r input.pcapng -T json > output.json
```

- This conversion generates a raw JSON file containing all captured data.



9. **Mapping JSON Data**

- Compare the structure of the generated JSON file with the sample JSON file provided by AntShield.

- Map the keywords and data fields from the AntShield sample JSON to the corresponding fields in the generated JSON file.

- This mapping ensures that the final JSON file adheres to the format required by AntShield.

10. **Generating Final JSON File**

- After mapping, generate the final JSON file which contains all the necessary data as specified by the AntShield sample JSON.

- This final JSON file is the product, containing the captured, decrypted, and structured data ready for analysis.

Note The JSON data is initially in hexadecimal format.

11. **Dealing with Hexadecimal Data**

- During the JSON conversion process, note that some data remains in hexadecimal format.
- Post-process the JSON to decode any hexadecimal data into plaintext as needed for analysis.

By following these detailed steps, we systematically capture network traffic, decrypt it, convert it to a JSON format, and map it to the required structure, ultimately producing a JSON file ready for analysis of PII leaks.

Proof of Concept (PoC) for PII Monitoring App on Android

Background

The initial goal was to develop an Android app capable of monitoring other installed apps to detect and report any PII leaks, aiming to achieve this without rooting the device or using third-party proxies. However, due to technical constraints and Android security policies, it became necessary to root the device and use MITM proxy for effective traffic decryption and PII monitoring.

Procedure

1. **Initial Setup and Rooting**
    - Root the device using a reliable method (e.g., Magisk).
    - Install Magisk to manage root permissions and handle system-level certificate installations.
2. **Installing PcapDroid**
    - Download and install PcapDroid on the rooted device.
    - Open PcapDroid and configure target apps for monitoring.
    - Specify the storage location for captured PCAP files.
3. **Configuring TLS Decryption**
    - Access PcapDroid settings and enable TLS decryption.
    - Follow the in-app guide to set up the MITM proxy and install trust certificates.
    - Magisk handles the system certificate installation, ensuring all apps trust the MITM proxy.
4. **Capturing Traffic**
    - Activate options for capturing as root and capturing the full payload.
    - Start PcapDroid to begin traffic capture.
    - Use the target apps normally to generate network traffic.
    - Stop the capture once sufficient data has been collected.
5. **Saving and Analysing Data**
    - Save the captured data as a PCAP file along with the ssllogkey.txt for TLS decryption.
    - Use Wireshark to load the PCAP file and apply the ssllog.key to decrypt TLS traffic.
    - Convert the PCAP file to PCAPNG format, combining it with the SSL log key.
    - Convert the PCAPNG file to a JSON format using Tshark.
6. **Data Analysis**
    - Analyse the JSON file to identify any PII leaks. Note that data in the JSON file may still be in hexadecimal format.
    - Perform keyword mapping to match AntShield's sample JSON file for consistent reporting.
    - Identify PII leaks by looking for unencrypted (plaintext) transmission of sensitive data.

Conclusions

- **Technical Necessities**: Rooting and using MITM proxies are necessary to decrypt and analyse app traffic for PII leaks due to Android's security policies.
- **Target Audience**: The current implementation is best suited for tech-savvy users who can perform rooting and manage certificates.
- **Future Directions**: To make the solution more accessible, the app could be pre-installed as a system application by phone manufacturers, eliminating the need for user-level modifications.

Final Note

- A PII leak occurs when sensitive information is sent over the network without encryption, meaning it can be read as plain text. Our initial goal was to develop an app that could monitor other Android apps for PII leaks without needing to root the device or use third-party proxies. However, we found that this was not possible due to Android's security policies. As a result, we had to use rooting and third-party proxy to decrypt and analyse the data.
- This means that currently, our app requires users to have technical knowledge about rooting and certificate installation. In the future, to make the app more user-friendly, it would be beneficial if phone manufacturers included it as a system app on their devices. This would remove the need for users to perform rooting and install certificates themselves, making it easier for everyone to use.