

S.C.O.U.T the Spyware Checking and Observation Utility Tool

**Investigation of technological assistance for the detection of
spyware with a focus on intimate partner surveillance**

Master thesis

for attainment of the academic degree of

Master of Science in Engineering (MSc)

submitted by

Tobias Etzenberger, BSc.

is221815

in the

University Course Information Security at St. Pölten University of Applied Sciences

Supervision

Advisor: FH-Prof. Dipl.-Ing. Peter Kieseberg

Assistance: -

Declaration

Title: S.C.O.U.T the Spyware Checking and Observation Utility Tool

Type of thesis: Master thesis

Author: Tobias Etzenberger, BSc.

Student number: is221815

I hereby affirm that

- I have written this thesis independently, have not used any sources or aids other than those indicated, and have not made use of any unauthorized assistance.
- I have not previously submitted this thesis topic to an assessor for evaluation or in any form as an examination paper, either in Austria or abroad.
- this thesis corresponds with the thesis assessed by the assessor.

I hereby declare that

- ☒ I have used a Large Language Model (LLM) to proofread the thesis.
- ☒ I have used a Large Language Model (LLM) to generate portions of the content of the thesis. I affirm that I have cited each generated sentence/paragraph with the original source. The LLM used is indicated by a footnote at the appropriate place.
- ☐ no Large Language Model (LLM) has been used for this work.

Date

Signature

Kurzfassung

Smartphones spielen im Leben eines jeden eine zentrale Rolle, sei es um sich in der Stadt zurechtzufinden, mit Familie und Freunden in Kontakt zu bleiben, Online-Banking zu betreiben oder Arbeitsaufgaben zu erledigen. Sie bieten zahlreiche Vorteile, doch ihre integrale Rolle in unserem Leben birgt auch Möglichkeiten des Missbrauchs, die potenziell Schaden anrichten können. Die häufigste Art, dies zu tun, ist der Einsatz von Spyware, da unsere Handys über sehr sensible Informationen verfügen und mit all den in ihnen eingebauten Sensoren dazu verwendet werden können, die täglichen Aktivitäten von Personen genau zu überwachen. Diese Tatsache ist besonders gefährlich, wenn sie mit Gewalt in der Partnerschaft einhergeht, da der Angreifer höchstwahrscheinlich physischen Zugang hat und das Opfer oft nicht weiß, wie der Täter es überwacht. Selbst wenn sie Hilfe bei Sozialarbeitern suchen, ist es für diese recht schwierig, das Gerät nach bösartigen Apps zu durchsuchen, und die verfügbaren Tools basieren entweder auf Blacklisten oder erfordern umfangreiches technisches Know-how.

In dieser Arbeit untersuchen wir, wie wir möglichst viele Indikatoren für problematische Apps von einem Smartphone sammeln können, ohne zusätzliche Software darauf zu installieren. Darüber hinaus beziehen wir zusätzliche Daten ein, die die Verwendbarkeit unserer Ergebnisse erhöhen, und implementieren ein leistungsstarkes Machine-Learning-Modell, das verdächtige Apps automatisch kennzeichnet. Darüber hinaus führen wir ein Verfahren zur Erkennung von Bluetooth Trackern ein, da deren Nutzung in den letzten Jahren zugenommen hat. Anschließend bündeln wir unsere Erkenntnisse in einem plattformübergreifenden Tool namens S.C.O.U.T., dessen Hauptaugenmerk darauf liegt, für technisch nicht versierte Nutzer so benutzerfreundlich wie möglich zu sein. Unser Tool hat gezeigt, dass es möglich ist, Spyware aufzuspüren, ohne sich auf eine Blacklist von Anwendungen zu verlassen, indem nur die den einzelnen Anwendungen gewährten Berechtigungen verwendet werden.

Abstract

Smartphones play a central role in everyone's life, from navigating the city, staying in contact with family and friends, doing online banking, to managing work tasks. They offer numerous benefits, yet their integral role in our lives also presents opportunities for misuse, potentially causing harm. The most common way to do that is to use spyware, as our phones possess very sensitive information, and with all the sensors built into them, they can be used to accurately monitor someone's daily activities. This fact is especially dangerous when this is combined with intimate partner violence, as the attacker has most likely physical access and the victim is often not aware how their abuser is monitoring them. Even when they seek help from social workers, it is quite difficult for them to search the device for malicious apps, and the available tools are either based on blacklists or require extensive technical know-how.

In this thesis, we investigate ways to collect as many indicators for problematic apps as possible from a smartphone without installing additional software on it. Furthermore, we include additional data that increases the usability of our findings and also implement a powerful machine learning model that automatically flags suspicious apps. In addition, we introduce a procedure to detect any Bluetooth base tracker as the usage of them increased in recent years. We then bundle our findings into a cross-platform tool named S.C.O.U.T., whose main focus is to be as user-friendly as possible for non-tech-savvy users. Our tool has shown that it is possible to detect spyware without relying on a blacklist of apps by using only the permissions granted to each app.

Contents

1	Introduction	1
1.1	Contribution	2
1.2	Thesis Outline	2
2	Background	3
2.1	Permissions in Android	3
2.2	Spyware	3
2.2.1	Dual-use apps	3
2.3	Random-Forest machine learning model	4
2.4	Bluetooth	4
2.4.1	Received Signal Strength Indicator (RSSI)	4
2.5	Attacker and victim model	4
2.5.1	Attacker model	4
2.5.2	Victim model	5
3	Related Work	7
3.1	Technology usage in intimate partner violence	7
3.1.1	Spyware	8
3.1.2	Trackers	9
3.2	Resources available in the context of IPS	9
3.3	Detecting Spyware	10
3.3.1	Static analysis	11
3.3.2	Dynamic analysis	11
3.4	Existing Software	12
4	Methodology	13

5	Approach	15
5.1	Setup	15
5.1.1	Testing devices	15
5.1.2	Frontend	15
5.1.3	Software architecture	16
5.2	Android devices	16
5.2.1	Android Debug Bridge (ADB)	16
5.2.2	Device information	17
5.2.3	Users	18
5.2.4	Accessibility	18
5.2.5	Accounts	19
5.2.6	Packages	19
5.2.7	Permissions	20
5.2.8	SHA-256 hash	22
5.2.9	Root	22
5.2.10	Random Forest model	23
5.3	Additional data	24
5.3.1	Permission details	25
5.3.2	Install permission classification	25
5.3.3	Runtime and appops permission categories	25
5.3.4	Installers	25
5.4	Bluetooth trackers	26
5.4.1	Scanning	26
5.5	Limitations	27
6	Results	31
6.1	Data analysis workflow	31
6.2	Welcome page	31
6.2.1	Scan component	31
6.2.2	Export/Import component	32
6.2.3	Guides component	32
6.3	Android page	32

6.3.1	Analytics tab	32
6.3.2	Packages tab	33
6.3.3	Accounts tab	34
6.3.4	Accessibility tab	34
6.3.5	Infected sample	34
6.4	Bluetooth page	35
7	Discussion	45
8	Conclusion	47
8.1	Future Work	47
	List of Figures	49
	List of Tables	50
	Acronyms	53
	Bibliography	55

1 Introduction

With the rise of smartphones being a companion for everyday tasks, they have become a central role in people's lives. From navigating the city, keeping in contact with other people, to more critical areas, like being able to work or do mobile banking. Additionally, with the increase in features, usability, and comfort they also became a tool to control, monitor, and manipulate people. For example, politicians, activists, and journalists are getting unknowingly monitored by state secret services via their phones, and it did not take long for the private sector to also create various services to abuse these devices for their capabilities [1]. Because of this, smartphones have been increasingly be used in intimate partner violence (IPV) which created new terms like technology-facilitated abuse (TFA) or intimate partner surveillance (IPS) to categorize this development. Companies tailored to creating spyware like *mSpy*, *FlexiSpy* or *TheTruthSpy*, just to name a few, provide their services to everyone without a complicated setup. Their marketing is focused on parental control or employee monitoring rather than spying on your partner, even though their names appear much more frequently on third-party forums to invade someone's privacy without their knowledge [2][3]. Then there are also so called *dual-use* applications, which were created for a legitimate use case, such as parental control or GPS navigation, but can be misused for tracking an intimate partner. Furthermore, it is difficult for a victim or even staff in a women's shelter to easily identify if a smartphone has been infected with spyware. The lack of technological knowledge and the difficulty using existing tools are the biggest obstacles in tackling this problem. This thesis investigates the possibilities of automatically finding suspicious applications on smartphones without the need to install an additional app on the victim's phone or relying on pre-compiled blacklists. Furthermore, we acknowledge the need in this context to also detect physical Bluetooth based trackers like AirTags as their availability and popularity increased in recent years. We then focus our findings on developing a cross-platform software that can be used to detect spyware and Bluetooth trackers. To sum up, this thesis should answer the following questions:

- What IPS relevant data can be extracted from a smartphone?
- What data can be used to detect spyware without the need for a blacklist?
- How can Bluetooth trackers be detected without targeting a single vendor?

1.1 Contribution

The following are the key achievements of this thesis:

- Collecting relevant information from a victim's Android phone
- Automatically detecting malicious apps based on their permissions
- Visualizing the collected data with a user-friendly interface
- Creating a method to detect any Bluetooth tracker
- Bundling all achievements into a single cross-platform software named S.C.O.U.T

1.2 Thesis Outline

The overall structure of this document is as follows:

- **Introduction:** As discussed earlier, the introduction (referenced as chapter 1) provides an overview of the topic, challenges, motivation, and the scope of the research.
- **Background:** This section (referenced as chapter 2) covers necessary prerequisites and fundamental knowledge relevant to the subject.
- **Related Work:** The related work section (referenced as chapter 3) presents existing research and literature that pertains to our study.
- **Methodology:** In this part (referenced as chapter 4), we delve into the methodology employed in our research.
- **Approach:** The approach section (referenced as chapter 5) discusses the practical implementation of our approach.
- **Results:** In this section, we report the results of our experiments in the section labeled chapter 6.
- **Conclusion:** Finally, we draw conclusions and summarize the key findings and contributions of this thesis in chapter 8.

2 Background

In this chapter, we cover necessary prerequisites and fundamental knowledge relevant to our thesis. Furthermore, we define the attacker and victim model that we consider in our approach.

2.1 Permissions in Android

In the Android operating system, there are two types of permissions: runtime and install-time permissions. The install-time permissions are automatically granted, as they present only very little risk to a user's privacy or other apps. This includes, but is not limited to, full network access, the ability to run at system startup or to prevent the system from going into the sleep mode. Runtime permissions, on the other hand, are treated more carefully as they are categorized as *dangerous* and need manual user permission to be used [4]. Additionally, with the release of Android 10 in 2019 some runtime permissions are now tristate permissions or appops permissions, which means that these permissions can not only be allowed or denied, they can also be only allowed while the application is running in the foreground. Therefore, for example, the usage of the microphone or tracking the location is only possible when the app is visible to the user [5].

2.2 Spyware

Spyware is a type of malware used for spying and collecting of personal information. It tries to stay hidden on the infected device, while still accessing hardware components like the camera or microphone of a smartphone. In addition, spyware collects all kinds of data depending on its target [6].

2.2.1 Dual-use apps

Dual-use apps are apps that were designed for a legitimate use case, such as Google's *Find My Device* or Apple's *Find My* app, but also any child safety or anti-theft app available. They can easily be repurposed for IPS as they usually work hidden in the background and provide privacy-related information like location, call logs and much more. Their danger lies in being available from the official app stores, which makes

them harder to detect, but often lack more sophisticated features as they are still restricted by the app stores policy [2].

2.3 Random-Forest machine learning model

The Random Forest, or random decision forest, is a machine learning model usually used for classification or regression tasks. It utilizes multiple decision trees during training to correct overfitting to the training set, which is the most common problem of decision trees [7].

2.4 Bluetooth

Bluetooth is a wireless technology standard that is used in short-range applications to exchange data between devices. It operates using radio waves in the 2.4 GHz frequency range and is usually available in every smartphone, laptop, and smartwatch.

2.4.1 Received Signal Strength Indicator (RSSI)

The Received Signal Strength Indicator (RSSI) is used to measure the radio signal strength emitted by Bluetooth devices. The value is the power level being received by an antenna, which depends on the chipset used. Therefore, the values cannot directly be mapped to a physical distance. Furthermore, the values are largely influenced by the environment and need to be filtered before use by collecting multiple readings. [8]

2.5 Attacker and victim model

Defining the attacker and victim model is an important task before approaching our thesis. These define the assumptions about the capabilities of an attacker and victim of IPS.

2.5.1 Attacker model

To consider is that the attacker's technological skill level is moderate. They are not hackers, so 0-days or any other advanced backdoor does not need to be considered, but tech-savvy enough to do online research and use publicly available software and apps. They are willing to spend a considerable amount of money to reach their goal on hardware and software. Furthermore, they also have access to their victim's belongings, as in an IPV context, we can assume that they are living together or sharing children. So they could install

malicious apps on an existing smartphone, gift a prepared smartphone to their victim, and also place trackers in their victim's car, bag, coat, or any other possible location.

2.5.2 Victim model

The victim is often dependent emotionally and economically on the attacker. They also typically exhibit low self-esteem, are insecure, and are submissive. Their technological skill level can be considered low, and with the ongoing abuse, they also regularly have self-doubt and loss of confidence, which makes it easier for the attacker to convince the victim that they do not spy on them. Most likely, they will also not recognize signs of manipulation on their devices, like special symbols appearing or unexpected changes in system behavior. Additionally, the constant monitoring and control the attacker has over them leads to increased fear and anxiety, which also influences the victim's judgment on that matter.

3 Related Work

In this chapter we highlight the current state of research about technology, and further spyware usage and countermeasures in the context of IPV.

3.1 Technology usage in intimate partner violence

IPV consists of physical or psychological abuse by a current or prior intimate partner. In recent years, the usage of technology drastically increased, and the so-called TFA is now more common for domestic violence (DV) specialist providers [9]. The authors of "Exploring the Impact of Technology-Facilitated Abuse and Its Relationship with Domestic Violence" [10] surveyed 15 agents from frontline DV organizations, which all confirmed that TFA is used in many cases. They stated that TFA is just another form of control, but makes it much easier and continuous to harm their victims as they now do not need physical presence anymore. Furthermore, they found that IPS is especially harmful, as victims may not be aware of this abuse or just do not know how their abuser was able to find them. In addition, survivors of TFA stated that they felt they could never escape their abuser, even after leaving the relationship, which is increasing the level of fear of having no way out of this abuse or no privacy anymore. Another way of monitoring a victim is through their children. They stated that an abuser targeted a victim's child by installing tracking software on the child's phone or sending messages to determine their current location, highlighting the necessity of checking children's devices. The authors of "Technology-Based Responses to Technology-Facilitated Domestic and Family Violence" [9] also looked into the countermeasures manufacturers have built into their products and found that they rule out some opportunities for abuse but are not preventing exploitation in general. This shows that even when companies try to design their products to mitigate TFA, it is very difficult to do so. In addition, workarounds for these mitigations will be faster found than these workarounds can be discovered and closed.

Year	Vendor	Accounts
2015	mSpy	699,793
2017	FlexiSpy & Retina-X	130,000
2018	SpyFone	44,109
2024	mSpy	2,394,179
2024	pcTattletale	138,751
2025	Spyic	875,999
2025	Spyzie	518,643
2025	Cocospy	1,798,059

Table 3.1: Data breaches of phone monitoring software vendors from 2015 to 2025. [12], [13]

3.1.1 Spyware

Abusers are nowadays more likely to use technology to monitor and control their victims. Recent data breaches of million accounts from different spyware vendors, which are listed in the following table 3.1, confirm this trend. The authors of "The Spyware Used in Intimate Partner Violence" [2] investigated multiple openly available spyware tools, where the majority are dual-use apps. They found that spyware is especially dangerous in the context of IPV, because installing these kinds of apps can be done by the attackers without the knowledge of their victim. This is the case, because they often know, guess, or enforce exposure of device credentials (password, pin, or swipe pattern). As many spyware apps are available on the app store, marketed as parental control software, installing them is as easy as any other app. Additionally, some spyware apps can only be side-loaded, which does not require deeper technical knowledge and most of the time also includes an easy-to-follow installation guide. Furthermore, rooting the phone is not required, but some more advanced features depend on it. There are also some vendors like *FlexiSpy* [11] who sell prepared smartphones with their spyware already installed.

Spyware Capabilities

The authors Chatterjee et al. [2] investigated the functionality of their collected malware samples and found that all apps that were not packaged by the Operating System (OS) need to be executed once after installation to run in the background. Additionally, the attacker needs to manually allow all requested permissions on the first execution, or else the app will not be able to access the required information (GPS, SMS, camera and call logs, etc.). There can still be dual-use apps pre-installed by the manufacturer or cellular provider,

which would have all required permissions already permitted. As an attacker normally has physical access to the victim's device, these special cases can be ignored. The researchers categorized spyware capabilities into three areas:

Monitoring. One crucial use case for an abuser is to know where its victim is currently (GPS), who the victim is talking to (SMS, call log), and what the victim is doing (photos, videos, app usage, or web history). Spyware can do all these things, and some apps are also capable of taking photos or recording audio on demand. Most dual-use apps, including the generally built-in 'find-my-phone' feature, use GPS for tracking the location data and sending it to a remote server. Other dual-use apps do not require an internet connection. Instead, they are triggered by an SMS and respond with the current location.

Secrecy. Spyware apps usually try to hide themselves, by not displaying an icon in the app drawer or any other place. Some apps are inaccessible from a victim's phone after installation, and others need a specific code dialed to reveal their settings.

Control. Most common for child safety software is to have remote control of the device. An attacker can block specific apps, impose browser restrictions, or limit the amount of time an app or the phone can be used in a day. It is also possible for them to remotely change device settings, like enabling Bluetooth or GPS.

3.1.2 Trackers

With the popularity of Apple's AirTags, the trend of using hidden surveillance devices in the context of IPS has also risen. The researchers Ceccio et al. reviewed different commercially available spy devices, detection devices, and detection apps in their paper [14] and found that there is currently a lack of useful and working detection systems while tracking devices are easy to get. They found that most of the spy devices collect video, audio, or location and communicate via cellular networks, Wi-Fi, or Bluetooth. Most of them are sold as dual-use devices for home, vehicle, children, or pet protection. Focusing on the GPS and Bluetooth trackers, they showed that GPS trackers have superior accuracy, but Bluetooth trackers are also sufficient to track someone's path, with the benefit of being very easy to obtain and use. Additionally, countermeasures of Bluetooth tracker manufacturers to stop their usage for IPS have already been circumvented.

3.2 Resources available in the context of IPS

Abusers have access to plenty of resources online, like guides in text and video form, blog posts, and advertisements from spyware vendors. Additionally, they often find assistance in public forums disguised as parental control, but also directly focusing on IPS. The authors of "The Web of Abuse" [3] looked into

the difference between available resources for attackers and survivors. In their survey from 63 individuals, they showed that 45% seek help online and nearly two thirds relied on public search engines. Their searches included legal and relationship advice, but also technical help to detect spyware. They tried to simulate these queries and found that many of the available websites relevant to mitigating IPS are not directly targeting victims. In addition, most of them do not recommend comprehensive tools or vigorous solutions, and some also contain bad or wrong information, such as using a Virtual private network (VPN) service or relying on anti-spyware apps, which often fail to detect dual-use apps. They came to the conclusion that this could also harm the victims, as this could give them a false sense of security if they failed to find spyware or trackers. Then they did the same research for the abuser side and found that more than half of their reviewed websites are directly focused on IPS using either spyware or dual-use apps. Furthermore, these websites are also a lot easier to understand and act upon than for victims.

The authors of "Technology-Based Responses to Technology-Facilitated Domestic and Family Violence" [9] showed that some websites provide information for victims of IPV, but also women in general that are concerned about their privacy. These websites include guides, tips, and strategies about specific threats in the TFA context. They found that the challenge these websites are facing is to keep their content up-to-date as platforms, applications, and devices will change in functionality and visuality overtime. Additionally, providing fully formulated instructions, for example, checking if spyware is installed on a device, will be so much content that it is unlikely to be consumable by their target group. They came to the conclusion that this circumstance increases the need for technology-based detections to help with the overwhelming amount of threats TFA is bringing.

3.3 Detecting Spyware

There are many papers around detecting malware in smartphone apps. Most of the static methods focus on app permissions retrieved from the corresponding manifest file. The researchers Ehsan et al. [15] looked into 16 different papers that focus on detecting malware with different machine learning classifiers. The most common classifier used in their reviewed paper was the Random Forest classifier and everyone used a different set of datasets. Most of them focus on static analysis, and only three of them used dynamic analysis in conjunction with it. They found that several proposed methods resulted in an over 90% accuracy level, but also hinted at a high chance of false positives and false negatives.

3.3.1 Static analysis

The paper "You are what the permissions told me! Android malware detection based on hybrid tactics" [16] demonstrates the usage of a deep learning text classification model TextCNN to detect malware by focusing on the permissions specified in an extracted AndroidManifest.xml file. With this model, they achieved an accuracy of 99.8%, which was 3.4% higher than the next best classifier. In addition, they used a dynamic feature to identify the malware family of the detected threat.

The authors Amer presented in their paper [17] an ensemble-based voting model that combines multiple classifiers to provide even better accuracy than an individual classifier. In their testing, they reached a 99.3% accuracy when combining their 5 classifiers, with RandomForest and Multilayer Perceptron Neural Network being the closest ones to reaching 99% by themselves.

In the paper "SF Droid Android Malware Detection using Ranked Static Features" [18] they not only based their static features on permissions, but also included intents, hardware components, content providers, services, and broadcast receivers. The authors found that by using only permissions when training their RandomForest model, they achieved an accuracy of 94.49%, which can be increased to 95.90% when all collected features are used.

The authors Yerima et al. [19] did also extract additional features in addition to the app permissions. They include intents, API calls, the existence of dangerous Linux commands, and check the presence of embedded files. They proposed "DroidFusion" which is based on four ranking-based algorithms to combine classifiers without traditional training methods. They demonstrated that they were able to improve performance by around 1% using both nonensemble and ensemble based classifiers and reached an average accuracy of 97%.

3.3.2 Dynamic analysis

The authors of "Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning" [20] acknowledged that labeled datasets are difficult to generate and therefore, proposed a semi-supervised machine learning model (PLDNN) to overcome this issue. In addition they created a new dataset called *CICMalDroid2020* that includes common static, but also dynamic features like: system calls, binder calls, composite behaviors, and network traffic. With their proposed model they were able to reach an accuracy of 97.84% which is considerably higher than their compared state-of-the-art semi-supervised machine learning model.

The paper "Towards a Network-Based Framework for Android Malware Detection and Characterization" [21]

focuses on only using network traffic to detect the presence of malicious apps on a smartphone. They were able to achieve an average accuracy of 91.41% for five different machine learning classifiers using nine flow-based, packet-based, and time-based network traffic feature measurements. In addition, they showed that it was also possible to identify the category of the detected malware.

3.4 Existing Software

The authors of Technology-Based Responses to Technology-Facilitated Domestic and Family Violence [9] looked into two tools for combating IPS. The first tool is hosted by the *Clinic to End Tech Abuse* [1] and scans a phone for known spyware and dual-use apps using a compiled blacklist. The second one, *TinyCheck* was developed by Kaspersky and acts as a WiFi access point that monitors the traffic for data going to suspicious servers, which are also based on a blacklist. At the time of writing the GitHub repository of *TinyCheck* (<https://github.com/KasperskyLab/TinyCheck>) is not accessible anymore. They both have the same limitations, as they depend on an up-to-date blacklist of apps, Domain Name System (DNS) names, or IP addresses. Furthermore, they still need a respectable amount of technical expertise to be used, as it is still not an easy task for a user to verify the categorization of legitimate apps and spyware with their data.

The authors Kumar et al. [22] created a tool *FAMOUS* that does not rely on pre-compiled blacklists. In their approach, they connect an Android phone to a computer, which pulls the apk file of all installed apps from it. Then they extract the *AndroidManifest.xml* from each apk to collect the permissions declared. Furthermore, they trained a Random Forest model on a weighted score-based feature set to detect malicious apps based on their permissions. With this approach, they were able to reach an accuracy of 99% in their testing, which outperformed their compared models.

4 Methodology

In this thesis, a new software was developed that should help victim's of IPS and staff in a women's shelter to identify spyware or physical trackers. It should not rely on a pre-compiled blacklist of apps and be usable on Windows, Linux, and macOS. For this, the following steps were done:

- **Data extraction:** The data needs to be extracted from the device, without installing any software directly on the victim's phone. This is important to not be detected by spyware, as this could alert an attacker and be dangerous for the victim.
- **Data analysis:** The extracted data needs to be automatically analyzed in order to detect if spyware could be present. This is done by linking specific data points to potential suspicious behavior and using a pre-trained machine learning model. Additionally, apps that use the Global Positioning System (GPS) of the smartphone should be treated special, as this is often used by spyware. Furthermore, detecting if the data extracted can be trusted is done by checking for a custom ROM and identifying if the device has been rooted.
- **Tracker detection:** Bluetooth-based trackers periodically broadcast data to keep track of their location in their network. Our first approach was to calculate the distance between each device to the scanner based on its signal strength. In our testing, we found out that this is very inaccurate when only using one scanner, as each device broadcasts with a different transmission power, therefore the distance measurements can vary significantly. Thus, we have developed a process that utilizes the change in signal strength when a device changes its distance to the scanner.
- **Presentation:** The data will be displayed in a cross-platform interface, which shows the data in a way that is easy to understand and act upon. It includes all data that gets collected to allow trained persons to analyze the data even further. It also provides easy-to-use searching capabilities in other online services and allow the user to uninstall suspicious apps directly via the interface.
- **Exportable:** The data is easily importable and exportable. For this, we use a common database system, which allows sharing the data remotely with social workers.
- **Evaluation:** To evaluate the software and its capabilities, we will scan two devices. One is an ordinary

smartphone that is actively used by a person. The other one utilizes a custom ROM and will be infected by spyware.

5 Approach

In the following sections, we explain in detail the tools and techniques we used in our research. Additionally, we explore and describe the data that has been gathered.

5.1 Setup

As the goal of this software is to be cross-platform, the programming language Python in the version 3.11 was used. This allows the software to run on any operating systems without the need to compile it. The software was tested on a Windows 10 machine with an external Bluetooth adapter plugged in, which scanned two smartphones and one tracker.

5.1.1 Testing devices

The first phone was a Samsung S24 with the Android version 14 that was actively used by a person, with the newest software version available, and without any malicious apps installed. The second one was an OnePlus 3 with the custom ROM LineageOS 18.1 installed that runs Android version 11. This device was infected with a version of the *Spynote* spyware found on MalwareBazaar [23]. These two Android versions were selected as they represent the upper and lower bound of the most used Android version at the time of writing, which made out 78.14% [24] of the worldwide market share.

5.1.2 Frontend

For the frontend, the following requirements were set: it needs to be cross-platform, have a responsive design, look modern, and in the best case is easy to work with. First, we looked at tkinter [25] as it is the oldest and most commonly used Graphical User Interface (GUI) framework for python. It is a robust but old library, so creating an eye-pleasing frontend with it was not an easy task. Therefore, Qt for Python with the package pyside6 [26] was considered. This is a well-rounded, modern, and cross-platform framework based on Qt 6, which was a bit too complex for our scenario. We then looked into NiceGUI [27] which is

a python package that was built for simple, but modern user interfaces running in a browser. It is build on top of FastAPI for the backend and utilizes Vue and Quasar for the frontend. This not only makes it cross-platform, but also enables the possibility to run on an external machine, like a Raspberry Pie. Additionally, it is very easy to work with as everything is written in code, including the UI, and in addition, is thread-safe by default.

5.1.3 Software architecture

The software consists of four main components: frontend, database, Android scan module, and Bluetooth scan module. This is visualized in figure 5.1. The frontend interacts with the other modules by either requesting the collection of data or retrieving collected data. As everything is written in python no additional middleware between the frontend and backend code is needed. The following sections further describe the scan modules in detail.

5.2 Android devices

Android devices account for two-thirds of the entire mobile operating system market [28]. In addition, Android phones can be configured to allow the installation of apps that can not found on the Play Store, which makes them an easier target to be infected with spyware.

5.2.1 Android Debug Bridge (ADB)

To collect any form of data, a connection to the smartphone is needed. For this the Android Debug Bridge (ADB) was used as it is available for every Android version since 2007. There are only two requirements. First, have the latest ADB version downloaded and installed on the computer. Second, the developer options need to be enabled, and USB debugging needs to be allowed on the phone. To enable the developer options, the Build number option needs to be found in the settings of the phone and then be tapped seven times, until the message "You are now a developer!" appears. Now the Developer options should be available in the system settings of the Android-based phone. [29]

Parsing

The output of ADB commands is in text form and not directly machine-readable, as the main target of ADB are human developers. To be able to parse the output data, we use the python package ttp [30]. TTP stands for *Template Text Parser* and was created to enable programmatic access to data produced by Command-line

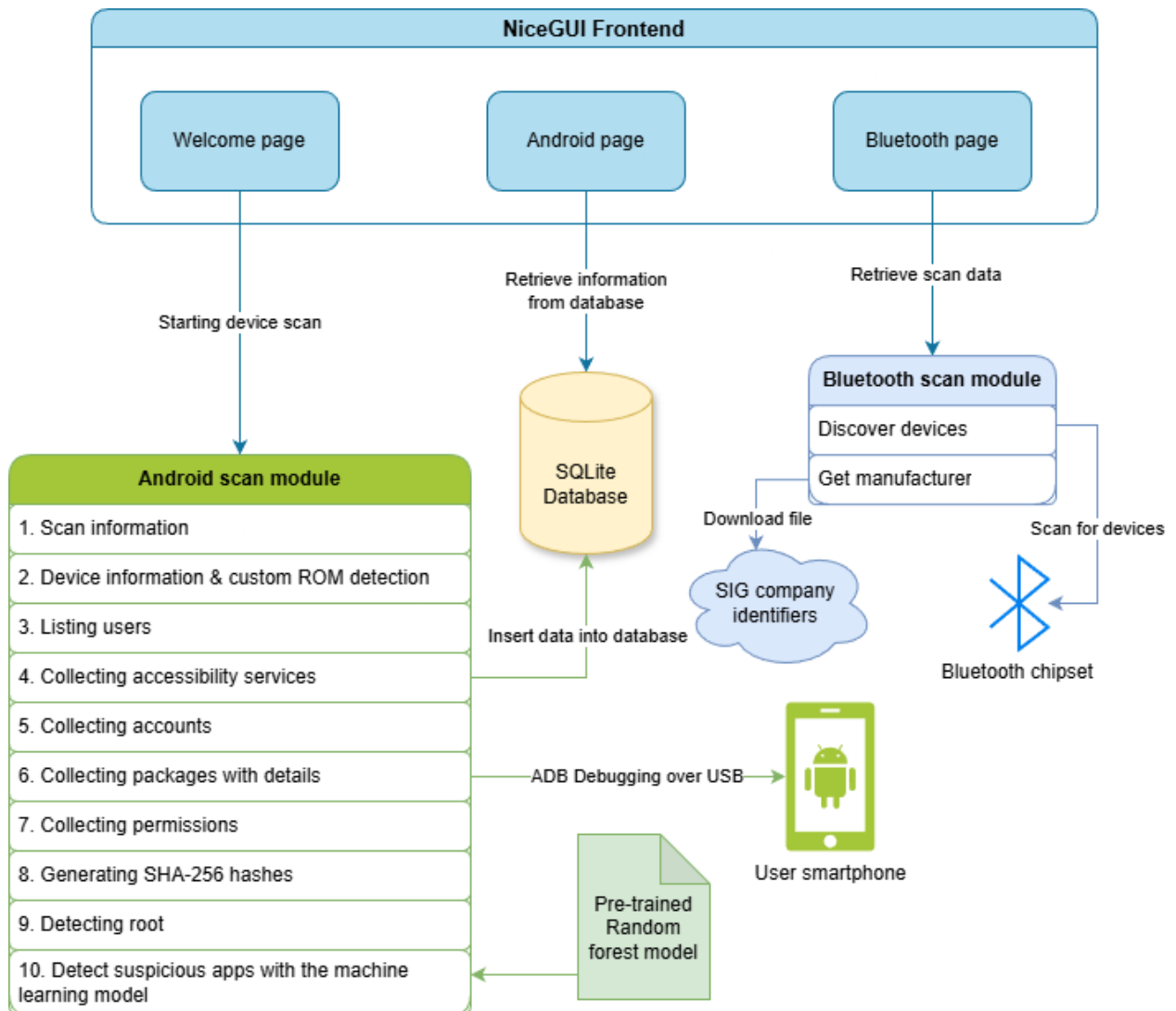


Figure 5.1: Software architecture of S.C.O.U.T

interface (CLI) of networking devices. It works by using two inputs, one that holds the data that should be parsed and the other that describes a template used for parsing the data. The template consists of template variables, normal text, and anchors to describe where the data is located and also supports a grouping system to create lists. This makes it a very powerful tool for parsing CLI output from ADB.

5.2.2 Device information

Information about the device can quickly give an overview, if the data extracted can be trusted. The data that was extracted is visualized in the following table 5.1.

Additionally, we try to detect if a custom ROM is installed on the device, instead of the official OS of the

Data	ADB command
Device brand	<code>getprop ro.product.brand</code>
Device model	<code>getprop ro.product.model</code>
Device serial	<code>getprop ro.serialno</code>
Android Version	<code>getprop ro.build.version.release</code>
Bootloader unlock status	<code>getprop sys.oem_unlock_allowed</code>
Boot image protection	<code>getprop ro.secure</code>
Active user	<code>am get-current-user</code>

Table 5.1: Device information and the corresponding ADB commands.

device manufacture. If this is the case, the data that was extracted can not be trusted, as it needs to be assumed that everything could be manipulated on the OS level. This check consists of two data points. First, the output of `getprop ro.build.tags` and secondly, the existence of `/etc/security/otacerts.zip` gets tested. If the output of the command is not *release-keys* or the file is not present, it proves that a custom ROM is used. [31]

5.2.3 Users

Android allows to have multiple users on one device by creating a user profile for each person. Each gets its own home screen, accounts, apps, and settings [32]. They distinguished themselves by a unique user ID; the first user is always named *Owner* and has the ID 0. This feature is mostly used to separate personal and work-related apps, but it is also used in other apps like Samsung's *Secure Folder*, which allows apps to be locked in a folder secured by a PIN code. The `pm list users` command is used to collect the following data for each available user.

- User ID
- Name
- Status

5.2.4 Accessibility

The accessibility settings are very sensitive, as an app that would have these privileges could: record voice calls (even from third-party apps), log keystrokes, take screenshots, prevent a system reboot, and many more [33]. To access this data, the following ADB command will be executed: `dumpsys accessibility`.

This will return all accessibility data from the phone. The important fields of this data are:

- Installed services
- Bound services
- Enabled services

The installed services include every possible accessibility service that could be enabled by the user. The bound services show which system event an accessibility service hooks, but does not need to list which app this belongs to. The enabled services, on the other hand, list which accessibility services are active and also which app they belong to. For this data, the `ttp` package could not be used, as the returning data changes depending on the accessibility services available. Therefore, a specific regex for every field was created instead.

5.2.5 Accounts

The accounts are very useful to check, because the victim may not know that there are accounts present that they are not aware of. For example, an attacker's Google account could be used for automatic backups, which could allow extracting data from the phone without any spyware on it. The data can be accessed via the `dumpsys account` command. Extracted will be all direct accounts and shared accounts. The direct accounts are the ones that are directly associated with an app, and shared accounts are direct accounts that have been shared with another app, most likely in a single sign-on scenario. Like when you do not have to log in again in Google Maps if you are already logged in into Google Mail. Data that can be extracted are as follows:

- User ID
- login name (direct)
- app (direct)
- login name (shared)
- app (shared)

5.2.6 Packages

All device apps are listed as a package in Android. As an example, the Play Store is the `com.android.vending` package. There are two ways to retrieve all available packages via ADB. One way is to use `pm list packages`, which can only show the packages of the current user. If another user is present and the ID of the current user is not supplied, this command returns with an error as it tries to access restricted data of other users. Therefore, we used the `dumpsys package packages` command, which does not only

return data from the current user, but also returns data from all other users present on the device. In addition, this command is very efficient as it not only returns additional useful data about each package, but also returns install and runtime permissions too. The package data that can be extracted are:

- ID
- Name
- APK folder path
- Version
- Installer package
- Installer UID
- Initiating package
- Originating package
- First install time
- Package flags

This data gives valuable information about a package, as the origin of a package could be an indicator of malicious activity. This provides information if any package was sideloaded and therefore does not originate from the official app stores. The originating package is an indicator of where the APK came from, for example, the used browser to download the APK file or the file browser used. Most of the time this returns null, as this is only set when an app gets side-loaded. Installer and initiating package is most of the time the same package; the only exception is when a package got installed via ADB, then the installer package is null. Interestingly, pre-installed system apps do not set any of them and can be identified by either having the installer UID set to -1 or including *SYSTEM* in the package flags.

5.2.7 Permissions

There are three types of permissions: install permissions, runtime permissions, and appops permissions. The following sections describe the collected data and its corresponding parsing.

Install permissions

The install permissions are automatically granted to an app and only have a minimal effect on the system or other apps, like using the internet, allowing to vibrate the phone, or managing accounts. There are two levels of permissions: normal and signature. The signature protection level only differentiates itself by needing the package to be signed by the same certificate as the app or the OS that defines the permission [4]. The data of the install permissions that can be gathered are:

- Permission
- Status

Runtime permissions

The runtime permissions are permissions that the user needs to allow and are not automatically granted. These permissions can also be categorized as dangerous as they handle more sensitive data and actions [4]. For example, the permission to use the camera, microphone, or access photos and videos needs to be granted by the user to be accessible from an app. Authorization only needs to be granted once, so an attacker could have confirmed the required permission and, the victim will never be asked for access to the camera again. The data that is extracted for every user are:

- User ID
- Installed status
- Hidden status
- Stopped status
- First install time
- Permission
- Status
- Flags

As it can be seen, not only permission information gets extracted, but also user-specific data about a package. This includes if the app is installed, if the app is hidden, and the first time the package was installed for the specific user.

Appops permissions

Appops permissions enable more fine control of runtime permission authorization. Normally, runtime permissions only have two states: allowed or denied, but appops introduces the foreground state. If the permission is only accessible in the foreground the app needs to be actively visible to the user to be able to use it [34]. This information is accessible with its own command `appops get {package}` and needs to be executed for every package. The resulting data does only belong to the current user and includes:

- Permission
- State
- Last used time
- Duration

- Last rejected time

This command brings two special cases up that need to be handled. First, the permissions *COARSE_LOCATION* and *FINE_LOCATION* belong to the GPS tracking and are documented everywhere else as *ACCESS_COARSE_LOCATION* and *ACCESS_FINE_LOCATION*. Therefore, their names have been edited so that they are more streamlined with other data points. Secondly, a permission can be reported twice when a state change has happened, thus the first entry printed is the current one. Additionally, the last used, duration, and last rejected time is provided in the following time format: `+..d..h..m..s...ms ago`, but can also include `(running)` to mark that this permission is currently in use. These special formats are parsed with the help of the current time of scanning to convert them into a datetime data type.

5.2.8 SHA-256 hash

To be able to check an app against VirusTotal or to verify integrity when an app needs to be extracted from the victim's phone, the SHA-256 hash gets calculated. This can only be done for the current user, and the `list packages -f -3 --user {user}` command is used to get all APK paths for all non-system apps. Then their SHA-256 hash is generated with the `sha256sum -b {location}` command.

5.2.9 Root

If a device is rooted, it grants apps superuser (root) access to the Android operating system. This allows to bypass restrictions of the manufacturer or carrier to system files and settings. In a spyware scenario, this can have tremendous security risks, as the spyware is now able to also bypass Android's sandboxing and read restricted data from other apps. Additionally, all other extracted data can not be trusted anymore, as it needs to be suspected that this has been manipulated. To detect if a device has been rooted, different areas on the Android phone can be checked. First, the existence of the following files and folders are tested:

- `/system/app/Superuser.apk`
- `/system/etc/init.d/99SuperSUDaemon`
- `/dev/com.koushikdutta.superuser.daemon/`
- `/system/sbin/daemonsu`
- `/sbin/su`
- `/system/bin/su`
- `/system/bin/failsafe/su`
- `/system/sbin/su`
- `/system/sbin/busybox`

- /system/sd/xbin/su
- /data/local/su
- /data/local/xbin/su
- /data/local/bin/su

Next, the availability of root-specific binaries are checked. They do not have to be able to be executed via ADB, their existence is sufficient.

- su
- busybox

Furthermore, all current running processes are collected and investigated if one of the following processes is present:

- supersu
- superuser
- daemonsu

Finally, typical root companion apps that indicate a rooted phone include:

- eu.chainfire.supersu
- com.noshufou.android.su
- com.koushikdutta.superuser
- com.zachspng.temprootremovejb
- com.ramdroid.appquarantine
- com.topjohnwu.magisk

If one of the checks has found something, the device is possibly rooted. [31] As it can be seen, these are all static checks, which means they rely on a hardcoded list of suspicious files, folders, binaries, processes, or packages. As this is a cat and mouse game, there will always be a way to circumvent these checks. Furthermore, keeping them up-to-date is a continuous task that is hard to do, especially if this is used in banking or other highly sensitive applications. To combat this, Google has provided the "SafetyNet Attestation API" which is now deprecated and superseded by the "Play Integrity API" which can dynamically check if a device is unmodified. [35] This API is not available via ADB and would need a companion app to be programmed and installed on the device.

5.2.10 Random Forest model

Detecting malicious apps is not as easy as flagging specific permissions, as this would create too many false positives. To solve this problem, we use the machine learning model *Random Forest* for classification.

This model provides very good results on detecting malware based on allowed permissions across different datasets [22] [18] [17]. The criteria for the dataset in our use case were to have different types of malware, include permission data, being labeled, and to be as recent as possible. For this, we found four different datasets:

1. Dataset from the authors of "Machine learning classifiers for android malware analysis" [36]
2. DroidFusion dataset [19]
3. CICMalDroid 2020 dataset [20]
4. Kronodroid dataset [37]

The first dataset only included around 400 samples, and with the release of 2016, it was not recent enough. The second one from 2018 looked more promising as it includes 15.000 samples, with one-third of them being malware apps. The third one included also data from 2018, but contains about 2.000 samples more and consists of malware apps for the most part. The last dataset is the one we used in our thesis and can be downloaded via their GitHub page [38]. It was the most recent dataset we found at the time, including samples from 2008 to 2020 and containing data from nearly 80.000 apps, with half of them being malware samples.

Before we could train a model on this dataset, we had to do some preprocessing. First, we removed all columns that do not include permission data, except for the *Malware* column, which labels the row as malware or benign app. For this, we used our generated file from 5.3.1 and only kept the columns that had a matching name. By using the package *sklearn* [39] we split the dataset randomly into a train and test subset, with the test subset representing 20% of the complete dataset. Then we trained the Random Forest classifier on the data and with it tried to predict the labels of the test subset. We found that this resulted in a 95% accuracy, which is on par with the testing Guerra-Manzanares et al. have done in their paper [37] this dataset is based on. We then exported the trained model and used this with the permissions collected in section 5.2.7.

5.3 Additional data

Some extracted data requires more technical knowledge to interpret. Therefore, in the following sections we describe the additional data we have created to support the user.

5.3.1 Permission details

Details to all default permissions, like description and protection level, help the user to understand and analyze the extracted data. As there were, at the time of writing, no up-to-date resources available in a machine-readable format, we developed a simple parser to generate a Yet Another Markup Language (YAML) file from the official documentation [40]. Later on, we discovered the `pm list permissions -f -g` command that also provides this information. The description was added in the frontend as a tooltip for every visible permission and the protection level also gets displayed as it can be seen in figure 5.2.

Permission	Category	Protection
android.permission.BLUETOOTH_CONNECT		dangerous
android.permission.READ_PHONE_STATE		dangerous
android.permission.RECORD_AUDIO		dangerous

Allows read only access to phone state, including the current cellular network information, the status of any ongoing calls, and a list of any PhoneAccounts registered on the device.

Figure 5.2: Permission detail tooltip

5.3.2 Install permission classification

To help a user with evaluating the install permissions that are automatically granted to an app, we implemented the table of criticality categories from the paper "Uraniborgs Device Preloaded App Risks Scoring Metrics" [41]. Therefore, the collected permissions can be mapped to one of the following categories: Astronomical, Critical, High, Medium, or Low.

5.3.3 Runtime and appops permission categories

To categorize permissions that access sensitive information, the corresponding support site from Google [42] was parsed and saved as an individual YAML file. The content of this file is listed in listing 1. This is used help the user to identify relevant permissions more easily.

5.3.4 Installers

To flag suspicious installers, a YAML file has been created with its content listed in listing 2. This includes the name, package name, and a suspicious flag for each included installer. We covered the most common official and third-party app stores and marked them as unsuspecting. Then we included the two most common suspicious installers: manually installed APK files and apps installed with the use of ADB. In our

testing, we have seen that Samsung used the package *com.google.android.packageinstaller* and LineageOS used *com.android.packageinstaller* for manually installed apps, which made sense as LineageOS does not ship with Google apps as default even if they are system apps.

5.4 Bluetooth trackers

Bluetooth trackers are based on Bluetooth Low Energy (BLE) as it reduces the power consumption drastically. They work by broadcasting a unique identifier every predefined interval. Scanning devices like phones or specific devices pick these signals up and forward the unique identifier with the current location to, for example, a cloud service. Previous research focuses on calculating the distance to a device based on this information. In our testing, calculating the distance with only one Bluetooth dongle did not result in accurate estimations of the distance. This has probably something to do with the fact that Bluetooth devices broadcast with different output powers that they do not include in their data. Therefore, we introduce a procedure that does not require any information from a Bluetooth device besides its signal strength and is still able to detect any Bluetooth based tracker. The following steps need to be done in order to detect a tracker:

1. The person or object should be removed from all known Bluetooth devices.
2. A scan is done, which collects all Bluetooth devices and their signal strength in the area.
3. The person or object gets closer to or further away from the scanner, depending on the current position.
4. A second scan is done, which collects all Bluetooth devices and their signal strength again.

After this procedure the data is analyzed, and if a device has changed significantly in signal strength it can, with high confidence, be said that a Bluetooth device is still attached to the person or object. This procedure can be repeated multiple times as needed.

5.4.1 Scanning

For the scanning, the python package *bleak* [43] was used, as it provides a cross-platform asynchronous BLE client. Each discovery phase lasts five seconds, which is the default in *bleak*, and our tests have shown that this is sufficient. This scan is repeated 5 to 30 times, decided by the user, to create a sample size that is sufficiently stable. Furthermore, to reduce the impact of environmental noise and other sources of interference, outliers are filtered out by using Chebyshev's inequality theorem [44] detailed in 5.1. For this equation, the recommended inequality for $k=2$ is applied as stated in the paper "Improving Distance Estimation in Object Localisation with Bluetooth Low Energy" [45]. Moreover, the mean of these values is

calculated and assigned to the device MAC address. If the resulting value is lower than -80, the measurement of this device is dropped, as everything lower than -80 is too unreliable for our use case.

$$Pr(|X - \mu| \geq k\sigma) \leq \frac{1}{k^2} \quad (5.1)$$

Additionally, to help the user identify devices, as only the MAC address or name would be displayed, we try to also detect the company of a device. For this, the company identifier is extracted from the data of the scanned devices, which is a 16-bit value assigned by the Bluetooth Special Interest Group (SIG) to recognize and interpret the manufacturer-specific data correctly. They provide a complete list of all assigned identifiers on their Bitbucket [46].

5.5 Limitations

There are several limitations in our thesis and programmed software that we acknowledge.

First, accessing an iOS device via USB is currently only possible when using macOS or Linux [47]. As the goal of the software is to be cross-platform, iOS devices could not be supported. This means that victims that have an iPhone cannot use our software, besides scanning for Bluetooth trackers.

Another limitation is our sample size of two devices. Although they already cover a wide range of different data points that could hint to suspicious activity, this may not be enough to be statistically significant when considering the number of different devices and configurations that exist. For this, more testing with real victim devices would need to be done.

Furthermore, the user can only be informed of the package name, which cannot necessarily be associated with the name of the app on the phone's home screen. This information is not possible to get via ADB without downloading each APK and extracting this data with another tool. This has a direct impact on usability for less technically savvy users and is definitely an issue that needs more attention in the future.

Our software also may not catch every spyware currently active in the wild, either due to user error or technical problems like not being detected by our static checks or machine learning model. In addition, our software could also scare a victim when many false positives are detected, even though our machine learning model has an accuracy of 95%.

Currently, our software can also only detect trackers that utilize Bluetooth for their tracking. Trackers that only use GPS or Wi-Fi are not covered and would not be possible to be detected.

```
1 - category: SMS and Call Log
2 permissions:
3   - READ_CALL_LOG
4   - WRITE_CALL_LOG
5   - PROCESS_OUTGOING_CALLS
6   - READ_SMS
7   - SEND_SMS
8   - WRITE_SMS
9   - RECEIVE_SMS
10  - RECEIVE_WAP_PUSH
11  - RECEIVE_MMS
12 - category: Location
13 permissions:
14   - FINE_LOCATION
15   - COARSE_LOCATION
16   - BACKGROUND_LOCATION
17 - category: All Files Access
18 permissions:
19   - MANAGE_EXTERNAL_STORAGE
20 - category: Photo and Video
21 permissions:
22   - READ_MEDIA_IMAGES
23   - READ_MEDIA_VIDEO
24 - category: Package (App) Visibility
25 permissions:
26   - QUERY_ALL_PACKAGES
27 - category: Request Install Packages
28 permissions:
29   - REQUEST_INSTALL_PACKAGES
30 - category: Full-Screen Intent
31 permissions:
32   - USE_FULL_SCREEN_INTENT
```

Listing 1: The YAML file with the categorisation of permissions that access sensitive information

```
1 - name: 'Google Play Store'
2   package: 'com.android.vending'
3   suspicious: 0
4 - name: 'Samsung Galaxy Store'
5   package: 'com.sec.android.app.samsungapps'
6   suspicious: 0
7 - name: 'Huawei AppGallery'
8   package: 'com.huawei.appmarket'
9   suspicious: 0
10 - name: 'F-Droid Store'
11   package: 'org.fdroid.fdroid'
12   suspicious: 0
13 - name: 'Manually installed apk'
14   package: 'com.google.android.packageinstaller'
15   suspicious: 1
16 - name: 'Manually installed apk'
17   package: 'com.android.packageinstaller'
18   suspicious: 1
19 - name: 'Manually installed via adb'
20   package: 'com.android.shell'
21   suspicious: 1
```

Listing 2: The YAML file of Android installer packages and their suspicious flag

6 Results

The Spyware Checking and Observation Utility Tool (S.C.O.U.T) is the resulting software of this thesis. It can be accessed via a browser, or in our case, in a native window. The structure and features of it are described in the following sections. The source code and installation instructions can be found at <https://git.nwt.fhstp.ac.at/is221815/s.c.o.u.t>.

6.1 Data analysis workflow

The workflow diagram illustrated in figure 6.1 shows the process a user should follow when analyzing the scan data from our tool. The workflow is designed to ensure a systematic and efficient approach to analyzing the collected data and improve the user's ability to detect possible spyware.

6.2 Welcome page

The first page a user sees is the welcome page 6.2. It consists of three components:

1. Scan component
2. Export/Import component
3. Guides component

6.2.1 Scan component

This component can have four different states 6.3. Every second, it is checked if ADB is installed, a device is connected, or if a device has authorized this computer for USB debugging. If more than one phone is currently connected, the user can also select the device they want to scan. Additionally, the name of the scanned person or phone can be provided, which gets saved into the database to help to connect the file to a person if this data gets exported.

When a phone is selected and authorized for scanning, the scan button can be clicked, and a modal with a progress bar and the current scan task is displayed 6.4.

6.2.2 Export/Import component

As the data is stored in a single SQLite database, the database file can be exported or imported. This allows a user to review old scans, compare them with a more recent one, or share the data with a service worker over the internet. Only one database file can be imported at a time, which, after successful loading, opens the android page 6.3.

6.2.3 Guides component

As we cannot assume that the user can research and also understand preparations they need to do in addition to installing this software, we have included the most common task directly into the interface. Therefore, this component provides step-by-step guides that a user can follow to help them with known challenges. At the time of writing, the installation of ADB on a Windows machine and enabling USB debugging on the phone are documented.

6.3 Android page

The Android page shows all data collected in an organized way. At the top, the scan, and device information with all device users are listed 6.5. This allows the user to quickly determine whether their phone has been tampered with. Furthermore, they are directly presented with the users present on the device and their current status. Below it are four tabs: analytics, packages, accounts, and accessibility, with analytics being pre-selected.

6.3.1 Analytics tab

This tab focuses on providing the user with analyzed data to help with finding problematic apps. First, the apps that are marked as suspicious by the Random Forest model and enabled accessibility services are shown. This guides the user to prioritize these apps for further investigation. As it can be seen in the figure 6.6 the app *com.teamspeak.ts3client* [48] has been flagged based on its permissions. In this case, this is a false positive as this is a legitimate voice chat and communication app, which is easily concluded by looking it up in the Google Play Store.

After that, an overview of all app installers is given. This visualizes from where apps are getting installed, which excludes all pre-installed system apps. In the figure 6.7 it can be seen that most of the apps were installed from the Google Play Store and Samsung Galaxy Store. Additionally, the F-Droid Store and also manually or over ADB installed apps were found. To help with categorizing, the apps installed by all

unknown or suspicious installers are listed. Suspicious installers like manual or ADB installed apps are highlighted with a red background to further signal the user that these apps should be looked at.

As location tracking can be very dangerous, but also very common in an IPS context, the apps that used the location service in the last 24 hours are listed in a table. Furthermore, we display the permission that is used to retrieve the location and the duration of the last usage, including an indicator if the app is currently using it. As it can be seen in figure 6.8, some permissions are occasionally used together when a location is tracked. This is very important as most dual-use apps focus on location monitoring, which may not be detected by the machine learning model.

Additionally, to the last 24 hours table, an overview of all apps that are allowed to use the location service is shown, as it can be seen in figure 6.9. The bar chart visualizes how many apps are always, or only while in the foreground, allowed to use the specific permission. Below that, a table of the underlying data can be used to filter for specific values. This will help the user to get an overview of how many apps use the location service and also which ones to investigate further, for example, filtering for all apps that are always allowed to use the location service.

6.3.2 Packages tab

The first table on the packages tab includes all packages with their version number, and they can also be filtered to not show pre-installed system apps. In the figure 6.10 we selected *com.teamspeak.ts3client* as it was flagged as suspicious in figure 6.6. Right next to it, its information is listed with the installation timestamp, from which app it got installed, and the sha256 hash. Below this, four blue buttons are available that can help the user identify the app by using its package name. The Play Store button uses the public Application Programming Interface (API) to list the information directly embedded into the interface. The rest open a browser window with the package name or hash already queried on the mentioned websites, as there is no public API available that does not need authentication for these websites. This has been implemented to help the user with further research, as we assume that the user does not have knowledge of these sites, nor does the user know how to use them with the information given by our software. Additionally, if the device is still connected, the selected app can also be uninstalled over ADB. This is sometimes the only way to get rid of a malicious package. To prevent accidental deletion of various packages, as it can be seen in figure 6.11, the user needs to solve a challenge before this is done. This is also the only feature of our software that actively modifies the user's phone. Also, a table of the information available per user is displayed. This shows which user has the app installed, when it was installed, if it is hidden, or if it has been stopped.

The next component shows the appops permissions, as it can be seen in figure 6.12, which shows if a permission is always allowed or only while the app is in foreground. As it can be seen, only the *RECORD_AUDIO* permission is available while in foreground for the selected package. Additional to that information, the time since it was last used, since it was last rejected, and the duration are displayed. These timestamps can be used by a user to detect if an app accessed information while in background, if it is known when the last time the app was actively opened. The protection level is visible, and the category a permission belongs to is also shown, as discussed in section 5.3.3.

The last two tables 6.13 provide information about the allowed install and runtime permissions. The category column for the install permissions shows a value from the categorization discussed in section 5.3.2. For the runtime permissions, the category is the same as for the appops permissions. It can be seen that the same permission can appear in one or multiple tables, but only appops include additional information.

6.3.3 Accounts tab

As it is important for a victim to see which accounts are used in various apps on their device, the login name, type (which is the referenced app), and the system user corresponding to this account are displayed. As it can be seen in figure 6.14 packages that do not allow having multiple accounts do not show the login name and instead just provide their app name. This data is only available when an app uses the built-in account manager, so most of the time this is more useful for dual-use apps.

As some apps support single sign-on with already existing accounts, this data is shown as cards to the user. In figure 6.15 it can be seen that the heading for each card is the login name of the account with the system user next to it surrounded by brackets. The packages that are using the provided account are listed below it.

6.3.4 Accessibility tab

On the last page, all accessibility data is displayed. As it can be seen in figure 6.16 there are some apps that provide accessibility services, but none are enabled. This means that some apps provide accessibility services, but neither of them are currently given any permissions to use accessibility functions.

6.3.5 Infected sample

To test the process with an infected device, the OnePlus 3 phone got a version of SpyNote [23] installed. In figure 6.17 we can directly see that this device has an unlocked bootloader and is also running a custom ROM, in our case LineageOS 18.1. This directly warns a user that their device has been probably tinkered with.

Our machine learning model correctly identified the installed app as malicious. Additionally, as this app is enabled as an accessibility service, it is also automatically flagged as suspicious. Furthermore, the user is also warned that this app has been installed manually. As it can be seen in figure 6.19 the package is an enabled accessibility service, but it also creates a bound service that does not provide the package it originated from.

6.4 Bluetooth page

The Bluetooth tracker that was used in our testing was a Flipper Zero [49] with the "FindMy Flipper" app [50] installed. This way, we successfully emulated an AirTag and other Bluetooth-based tracking devices. Each scan had a sample size of 10 scans and was repeated a total of 3 times. The output is shown in 6.20. As it can be seen, the Flipper Zero is present two times, once only with its name "Flipper Trik3n", as this is its default Bluetooth broadcast visualized by the dark green line. It is also visible as an AirTag without a name, which broadcasts with a MAC address of 11:22:33:44:55:66 and was correctly identified as an "Apple, Inc." device, visualized with the light blue line. The emulated broadcast used a higher transmission power of 6 dBm, which explains the gap between these two lines. For the first scan, the device was placed 1 meter away from the attached Bluetooth dongle, which resulted in a RSSI of -56.33. For the second scan, it was placed 5 meters further away, which resulted in a RSSI of -71.83 and then again placed directly on top of the dongle, which produced a RSSI of -31.67. As it can be seen, the movement of the dongle is directly represented by the graph, and if this device were hidden on a person, it would have been detected.

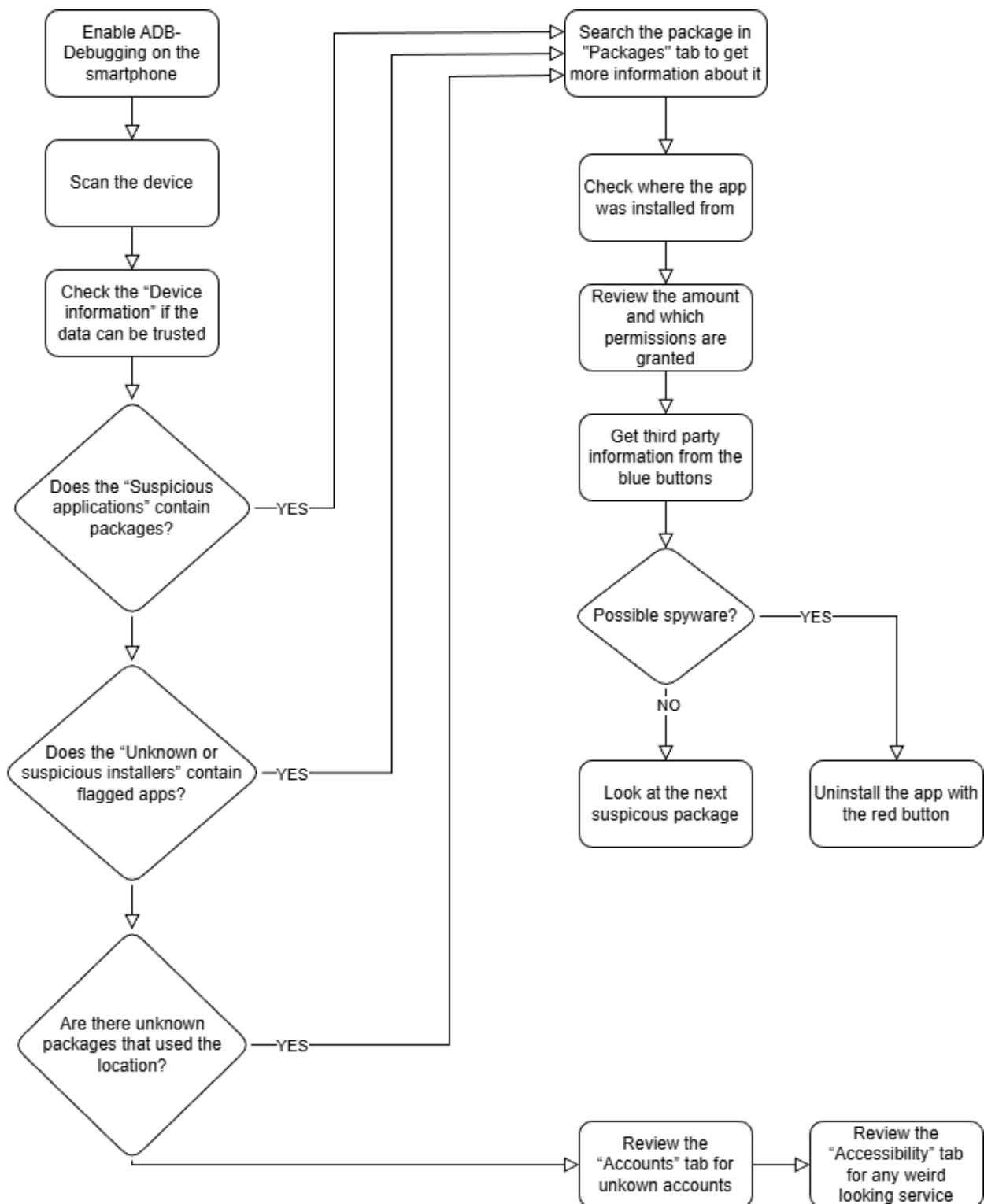


Figure 6.1: Workflow diagram of how the user should analyze the scan data for spyware

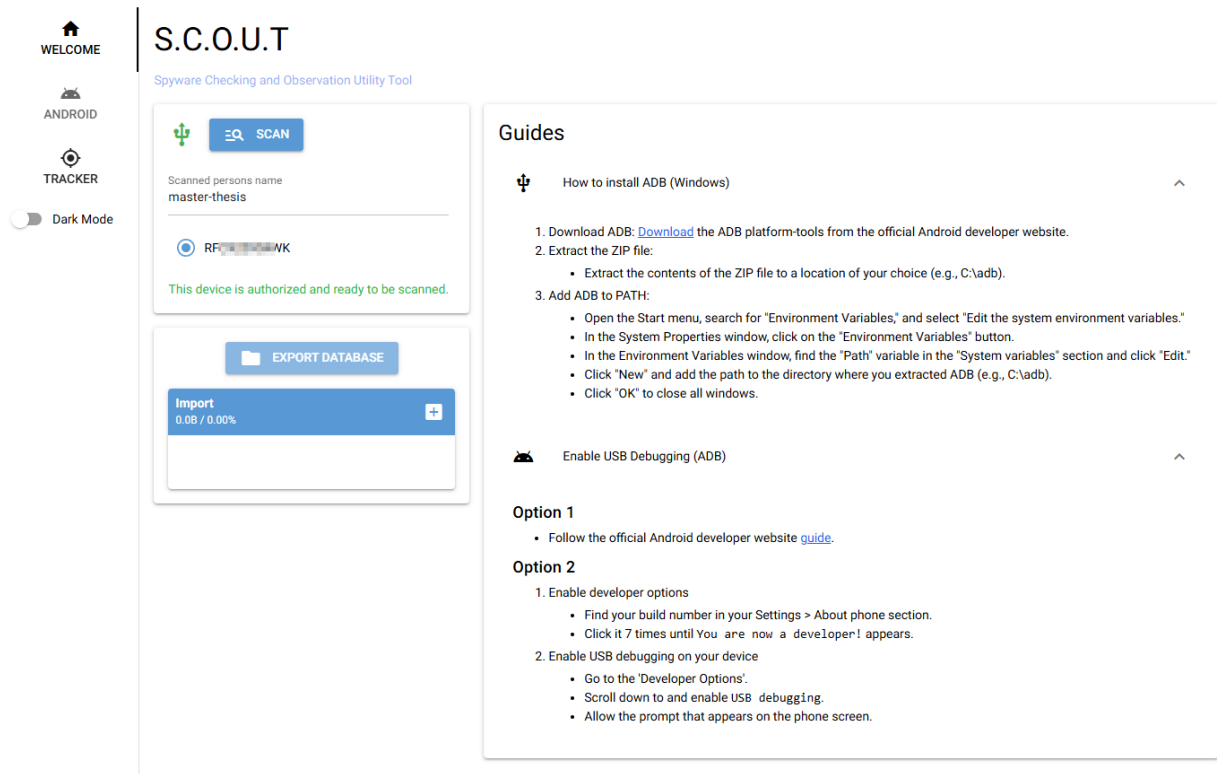
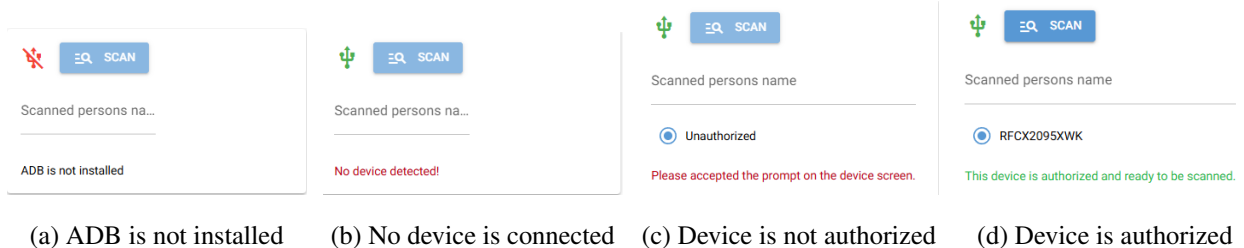


Figure 6.2: Welcome page, with the three components: scan, export/import, and guides.



(a) ADB is not installed (b) No device is connected (c) Device is not authorized (d) Device is authorized

Figure 6.3: The four states of the scan component that indicate if a device is ready for scanning.

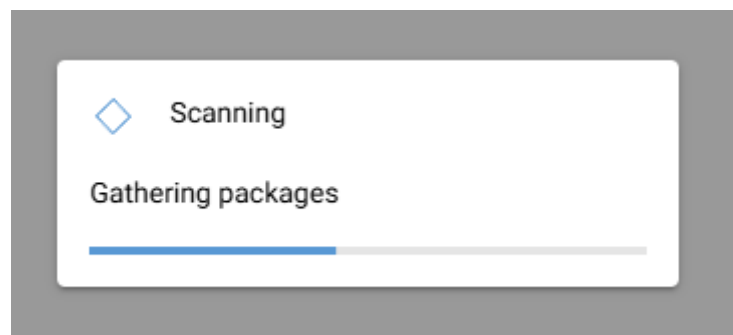





Figure 6.4: Scanning modal with progress bar and current task.

Android

Scan information		Device information		Device users			
Person	Tobias	Phone:	samsung [SM-S921B]	active	id	name	status
Scan version	1	Serial:	RF[REDACTED]WK	✓	0	Owner	running
Time of scan	2025-02-24 14:52:14	Android:	14	✗	150	Secure Folder	running
		Bootloader:	✓ (locked)				
		Boot image:	✓ (protected)				
		OS:	✓ (official)				
		Unrooted:	✓				

 ANALYTICS

 PACKAGES

 ACCOUNTS


 ACCESSIBILITY

Figure 6.5: Scan and phone information of the Samsung S24.

Suspicious applications		
Name	Installer	Source
com.teampeak.ts3client	Google Play Store	AI

Figure 6.6: Automatically detected suspicious applications of the Samsung S24.

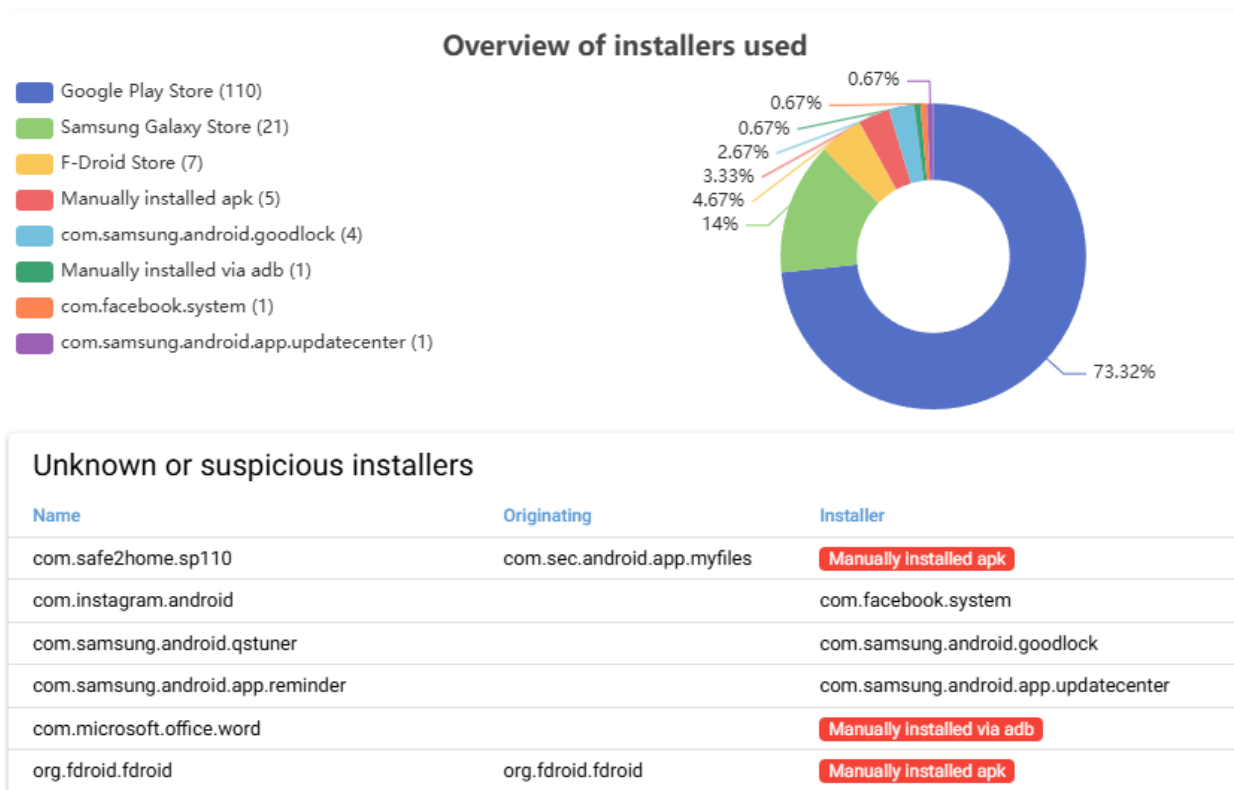
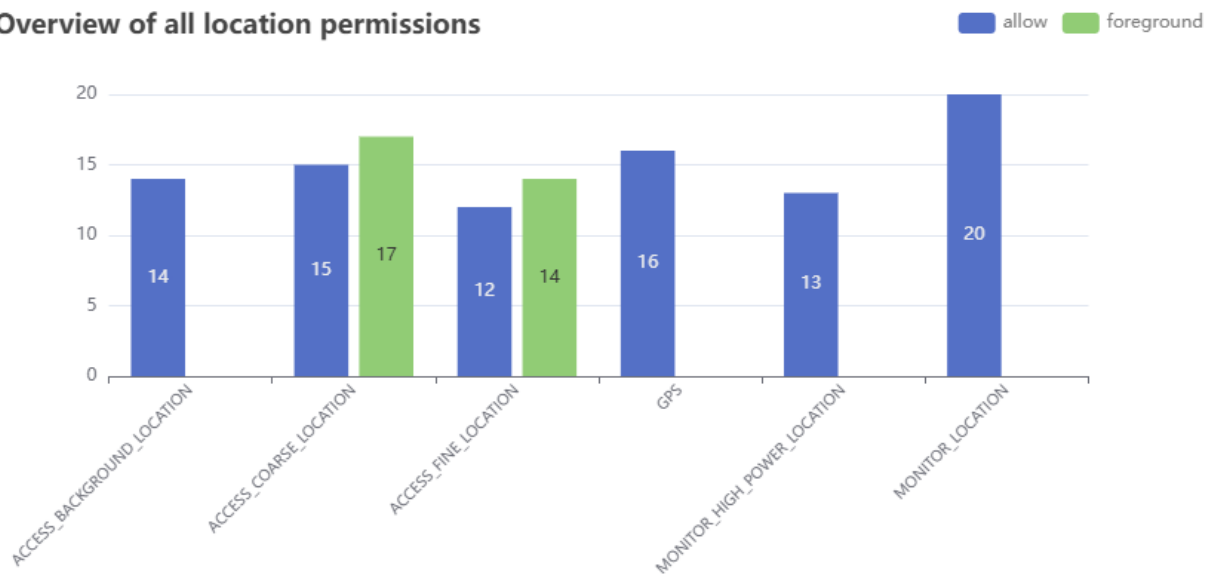


Figure 6.7: Overview of installers used to install apps on the Samsung S24.

Location tracked (last 24h)			
Name	Permission	Last_used_time	Duration
com.sec.android.daemonapp	ACCESS_FINE_LOCATION	23.02 17:38:32	
com.google.android.apps.maps	GPS	23.02 18:39:36	+1m59s315ms
com.google.android.apps.maps	MONITOR_LOCATION	23.02 18:39:36	+1m59s318ms
com.google.android.apps.maps	MONITOR_HIGH_POWER_LOCATION	23.02 18:39:37	+1m57s844ms
com.google.android.gms.location.history	ACCESS_FINE_LOCATION	23.02 18:40:57	
com.google.android.apps.maps	ACCESS_FINE_LOCATION	23.02 18:41:35	
com.samsung.android.mcfds	ACCESS_FINE_LOCATION	23.02 18:41:37	
com.samsung.android.rubin.app	MONITOR_LOCATION	24.02 02:27:12	+1s754ms
com.google.android.gms	MONITOR_LOCATION	24.02 02:27:12	+1s720ms
com.samsung.android.mcfds	MONITOR_LOCATION	24.02 02:27:13	+744ms

Figure 6.8: Apps that used the location service in the last 24 hours on the Samsung S24.

Overview of all location permissions



Name	Permission	Value	Last_used	Duration
<input type="text"/>	<input type="text"/>		<input type="text"/>	
com.sec.imsservice	GPS	allow	+367d21h5m8s285ms ago	+10s604ms
com.sec.imsservice	MONITOR_LOCATION	allow	+367d21h5m8s312ms ago	+10s576ms
com.google.android.apps.maps	ACCESS_COARSE_LOCATION	foreground		
com.google.android.apps.maps	ACCESS_FINE_LOCATION	foreground	+20h11m3s901ms ago	
com.google.android.apps.maps	GPS	allow	+20h13m3s1ms ago	+1m59s315ms
com.google.android.apps.maps	MONITOR_HIGH_POWER_LOCATION	allow	+20h13m1s713ms ago	+1m57s844ms
com.google.android.apps.maps	MONITOR_LOCATION	allow	+20h13m3s000ms ago	+1m50s318ms

Figure 6.9: Overview of all apps that are allowed to use a location permission on the Samsung S24.

Name	Version	System
teams		
com.teamspeak.ts3client	34.5	0
com.microsoft.teams	1416/1.0.0.2025032402	0

Package information	
Selected:	com.teamspeak.ts3client
InstallTime:	2024-08-23 21:09:45
Originating:	
Initiating:	Google Play Store
InstallerUid:	10260
Installer:	Google Play Store
Hash:	6ae9bb36842c4081aa25211ec73c3ef6d98c304b0bb4bc795d7ee1f0b8b7675c

PLAY STORE INFORMATION	SEARCH VIA APKPURE	SEARCH VIA GOOGLE	CHECK VIA VIRUSTOTAL	UNINSTALL APP
------------------------	--------------------	-------------------	----------------------	---------------

Affected user profiles				
User	Installed	InstallTime	Hidden	Stopped
0	1	2024-08-23 21:09:45	0	1
150	0	1970-01-01 01:00:00	0	1

Figure 6.10: Table of all installed packages, with specific package information and all affected users.

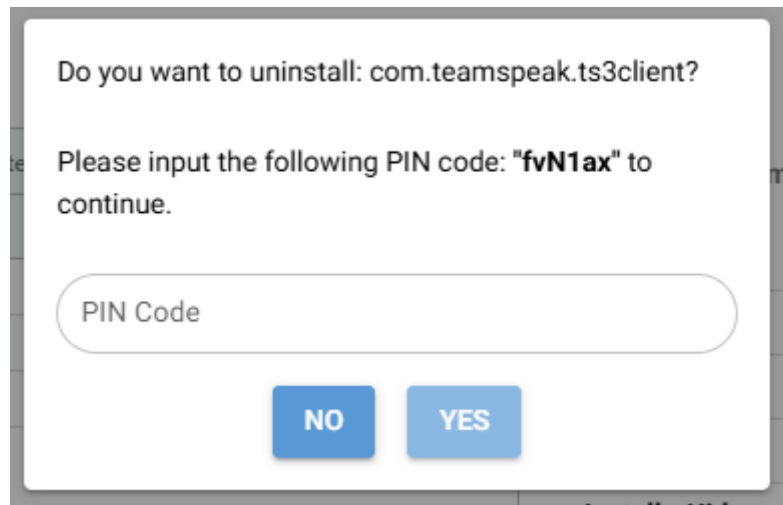


Figure 6.11: Uninstall modal with user challenge.

Appops permissions						
Permission	Value	Last_used	Duration	Rejected	Category	Protection
RECORD_AUDIO	foreground	+184d17h49m17s420ms ago	+1m42s8ms			dangerous
VIBRATE	allow	+184d18h41m20s129ms ago	+132ms			normal
AUDIO_MEDIA_VOLUME	allow	+184d18h40m47s921ms ago				
WAKE_LOCK	allow	+184d17h49m17s415ms ago	+1m41s941ms			normal
MUTE_MICROPHONE	allow	+184d17h47m33s377ms ago				
READ_PHONE_STATE	allow	+184d18h3m15s371ms ago				dangerous
START_FOREGROUND	allow	+184d17h47m20s972ms ago	+2d16h1m39s401ms			
MANAGE_EXTERNAL_STORAGE	default			+184d18h43m4s801ms ago	All Files Access	signature appop preinstalled
BLUETOOTH_CONNECT	allow	+184d18h43m2s852ms ago				dangerous
ACCESS_RESTRICTED_SETTINGS	allow	+184d17h47m20s885ms ago				

Figure 6.12: Appops permissions table from the package com.teamspeak.ts3client.

Install permissions				Runtime permissions		
Permission		Category	Protection	Permission	Category	Protection
android.permission.MODIFY_AUDIO_SETTINGS			normal	android.permission.BLUETOOTH_CONNECT		dangerous
android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS			normal	android.permission.READ_PHONE_STATE		dangerous
android.permission.INTERNET			normal	android.permission.RECORD_AUDIO		dangerous
com.android.vending.CHECK_LICENSE						
android.permission.BROADCAST_STICKY			normal			
android.permission.ACCESS_NETWORK_STATE			normal			
android.permission.VIBRATE			normal			
android.permission.ACCESS_WIFI_STATE		HIGH	normal			
android.permission.WAKE_LOCK			normal			

Figure 6.13: Install and runtime permission tables from the package com.teamspeak.ts3client.

Accounts available		
Login	Type	User
[redacted]@gmail.com	com.google	Owner
[redacted]@gmail.com	com.google	Owner
[redacted]@gmail.com	com.google	Owner
[redacted]@gmail.com	com.osp.app.signin	Owner
[redacted]@gmail.com	com.samsung.android.mobileservice	Owner
Signal	org.thoughtcrime.securesms	Owner
WhatsApp	com.whatsapp	Owner
Work account	com.microsoft.workaccount	Owner
is221815@fhstp.ac.at	com.microsoft.workaccount	Owner

Figure 6.14: Table of all accounts available on the Samsung S24.

Accounts shared with apps

<p>WORK ACCOUNT (OWNER)</p> <p>com.azure.authenticator</p>	<p>[redacted]@GMAIL.COM (OWNER)</p> <p>com.google.android.apps.maps</p> <p>com.google.android.apps.translate</p> <p>com.google.android.apps.tachyon</p> <p>com.google.android.gm</p> <p>com.google.android.setupwizard</p> <p>com.google.android.apps.restore</p> <p>com.google.android.apps.docs</p> <p>com.android.vending</p>	<p>IS191001@FHSTPAC.AT (OWNER)</p> <p>com.azure.authenticator</p>
<p>[redacted]@GMAIL.COM (OWNER)</p> <p>com.google.android.apps.maps</p> <p>com.google.android.apps.translate</p> <p>com.google.android.apps.tachyon</p> <p>com.google.android.setupwizard</p> <p>com.android.vending</p>		<p>IS221815@FHSTPAC.AT (OWNER)</p> <p>com.azure.authenticator</p> <p>com.azure.authenticator</p>

Figure 6.15: The shared accounts cards with their corresponding packages.

Accessibility services found			
Id	Type	Package	Data
1	installed	com.google.android.apps.accessibility.voiceaccess	.JustSpeakService (A11yTool)
2	installed	com.microsoft.appmanager	com.microsoft.mmx.screenmirroring.accessibility.ScreenMirroringAccessibilityService
3	installed	com.samsung.accessibility	.universalswitch.UniversalSwitchService (A11yTool)
4	installed	com.samsung.accessibility	.assistantmenu.serviceframework.AssistantMenuService (A11yTool)
5	installed	com.samsung.android.accessibility.talkback	com.samsung.android.marvin.talkback.TalkBackService (A11yTool)
9	installed	com.azure.authenticator	com.microsoft.brooklyn.module.accessibility.BrooklynAccessibilityService
10	installed	com.x8bit.bitwarden	.Accessibility.AccessibilityService

Figure 6.16: The table off all accessibility services found on the Samsung S24.

Scan information		Device information		Device users		
Person	Infected ROM	Phone:	OnePlus [ONEPLUS A3003]	active id	name	status
Scan version	1	Serial:	12fae9d2	✓	0	Owner running
Time of scan	2025-02-24 15:22:35	Android:	11			
		Bootloader:	✗ (unlocked)			
		Boot image:	✓ (protected)			
		OS:	✗ (custom rom)			
		Unrooted:	✓			

Figure 6.17: Scan and phone information of the infected OnePlus 3

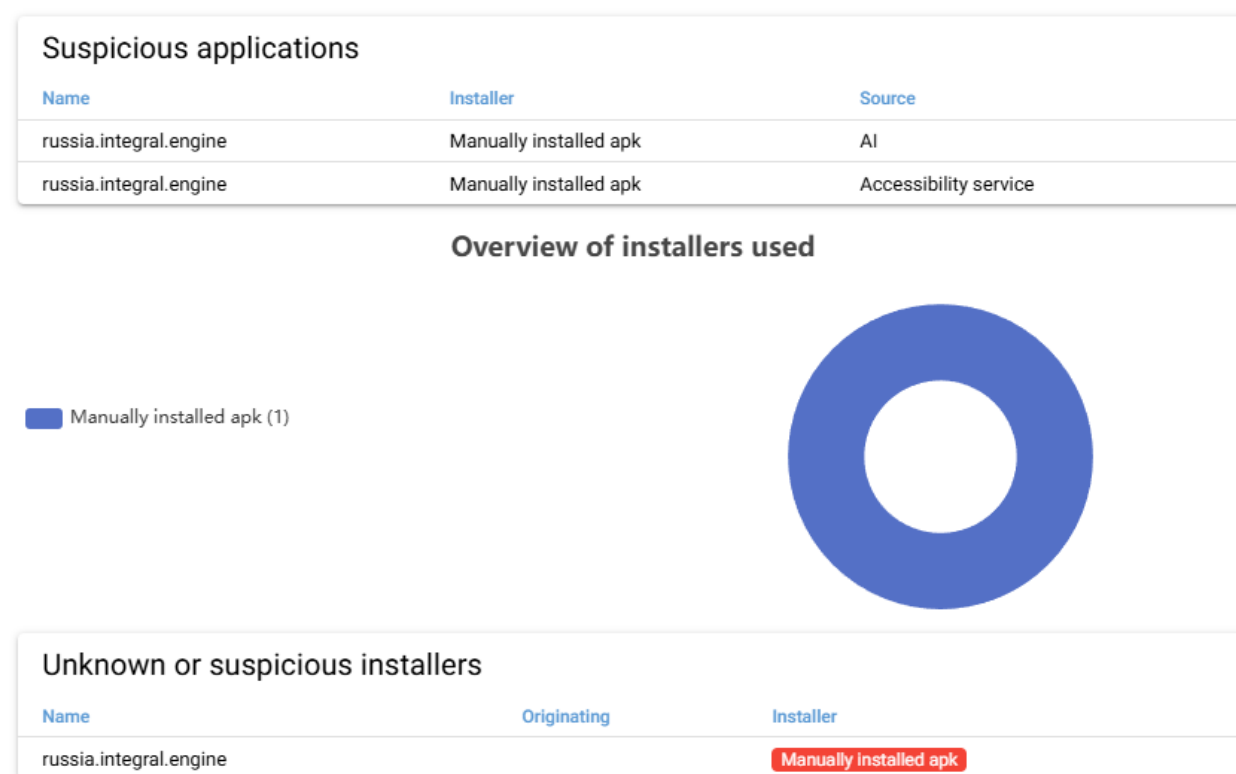


Figure 6.18: Suspicious apps and installers of the infected OnePlus 3

Accessibility services found			
Id	Type	Package	Data
1	enabled	russia.integral.engine	russia.integral.172
2	bound		Service[label=Chrome, feedbackType[FEEDBACK_SPOKEN, FEEDBACK_HAPTIC, FEEDBACK_AUDIBLE, FEEDBACK_VISUAL, FEEDBACK_

Figure 6.19: Accessibility services table of the infected OnePlus 3

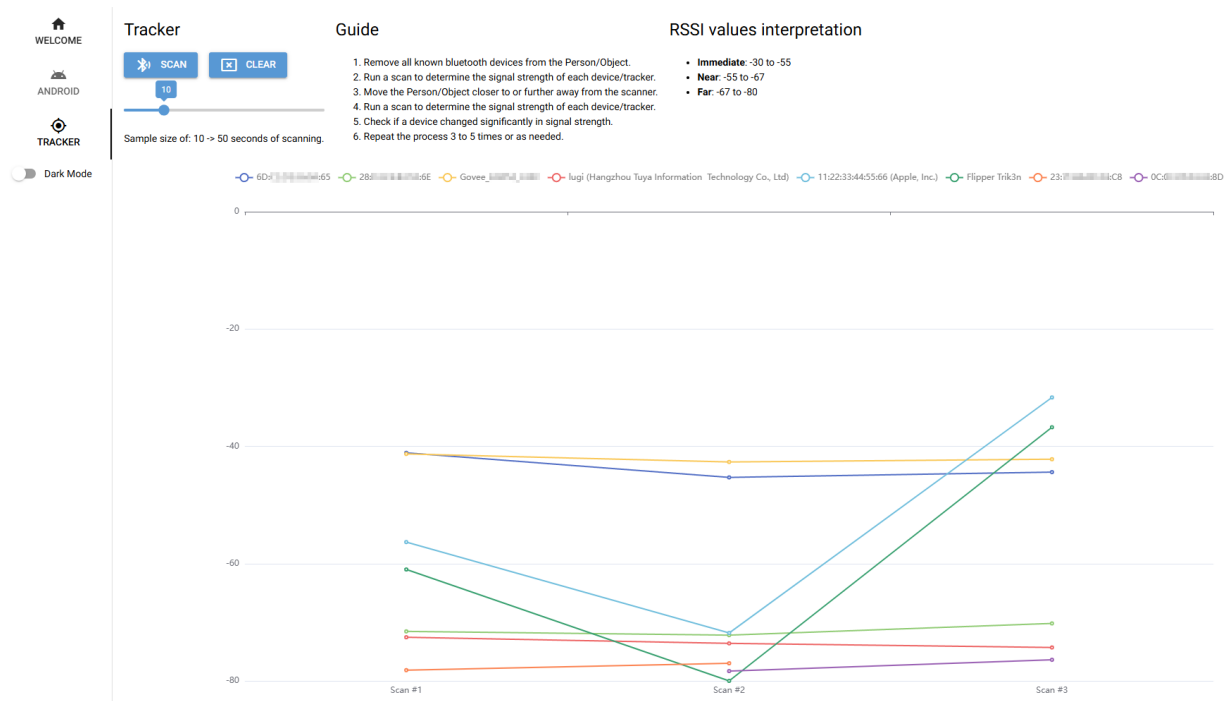


Figure 6.20: Bluetooth tracker detection graph after three scans, with the tracker being moved between the individual scans.

7 Discussion

With the increasing popularity of spyware and its critical impact on IPS, effective detection mechanisms and a user-friendly presentation are essential. To our knowledge, S.C.O.U.T is the first cross-platform software that does not only rely on package names to detect spyware. With it, we proved that creating such software is possible, even without the usage of blacklists that are labor-intensive to maintain. In the case of iPhones, there are currently no cross-platform tools available to access the needed information to detect suspicious installed apps. Furthermore, the existing Linux-focused library is an under active development community project [47] not associated with Apple Inc., which raises the question of why Apple does not provide such a tool in the first place. Users have to rely on their app stores to protect them from spyware, which does not help when dual-use apps are used instead. For Android devices, the tool ADB directly from Google can be used, which is able to retrieve all necessary information. As this information is more tailored to developers, the most difficult part is to do the processing of the data needed to be more understandable for less tech-savvy users. Here, manufacturers could help out by improving the data returned by their tools or providing better and more complete resources to interpret data. Currently, spyware is not monitoring USB activity and therefore not aware of the usage of our tool. When software like ours is used on a large scale, the developers of spyware may implement more sophisticated methods to also detect the usage of them. It remains to be seen how this will affect the reliability of the data we collect and the potential danger to a victim who uses it. The Bluetooth tracker detection was an interesting topic to tackle as most of the research and software available runs on the victim's phone, which in an IPS scenario can not be trusted. Here a simple but effective method was introduced to detect these trackers.

8 Conclusion

In this thesis, we investigated ways to detect spyware on Android devices by utilizing ADB and extracting the necessary information with it over USB. We described each command that was used with the resulting features and their use in detecting potential indicators of malicious behavior. Additionally, we introduce a method to detect Bluetooth trackers by analyzing the signal strength of all available Bluetooth devices. Furthermore, a cross-platform software named S.C.O.U.T has been developed, which was used to present the user with the collected information. The interface was tuned to help users analyze the data, even if they have less technical knowledge. For this, we split the information into different areas, with the first one focusing on crucial indicators for potential malicious apps. In addition, step-by-step guides, tooltips, and third-party information were built into the software to further improve the usability for a potential victim. We also used a machine learning model to automatically detect suspicious apps that was trained on a state-of-the-art dataset, which was able to reach an accuracy of 95%. Moreover, we talk about the limitations we were facing while developing this software. We then demonstrated that our tool was able to detect spyware on our manipulated smartphone, and also that we successfully identified a potential Bluetooth tracker.

8.1 Future Work

In order to continue the research of this thesis, additional features like intents, content providers, broadcast receivers, etc. could be gathered. This could, as already stated in other papers, increase the accuracy of the current machine learning model and further reduce the false positive rate. In addition, more machine learning models could be examined to further increase the efficiency of the automatic spyware detection. Another enhancement could be to include other data sources that improve the usability for non-technical users. For example, categorizing or ranking permissions to further visualize and explaining dangerous permissions in an IPS context could be improved. Building a standalone image for a Raspberry Pi could also be investigated, which could then also include the research of scanning iPhones for spyware. The introduction of this software in the women's refuge and the investigation of the problems that arise during its use would benefit the development of this and other similar software. Additionally, the detection of

8 *Conclusion*

Bluetooth trackers could be extended so that known tracker brands are also recognized by evaluating the data they transmit.

List of Figures

5.1	Software architecture of S.C.O.U.T	17
5.2	Permission detail tooltip	25
6.1	Workflow diagram of how the user should analyze the scan data for spyware	36
6.2	Welcome page, with the three components: scan, export/import, and guides.	37
6.3	The four states of the scan component that indicate if a device is ready for scanning.	37
6.4	Scanning modal with progress bar and current task.	37
6.5	Scan and phone information of the Samsung S24.	38
6.6	Automatically detected suspicious applications of the Samsung S24.	38
6.7	Overview of installers used to install apps on the Samsung S24.	39
6.8	Apps that used the location service in the last 24 hours on the Samsung S24.	39
6.9	Overview of all apps that are allowed to use a location permission on the Samsung S24.	40
6.10	Table of all installed packages, with specific package information and all affected users.	40
6.11	Uninstall modal with user challenge.	41
6.12	Appops permissions table from the package com.teamspeak.ts3client.	41
6.13	Install and runtime permission tables from the package com.teamspeak.ts3client.	41
6.14	Table of all accounts available on the Samsung S24.	42
6.15	The shared accounts cards with their corresponding packages.	42
6.16	The table off all accessibility services found on the Samsung S24.	42
6.17	Scan and phone information of the infected OnePlus 3	43
6.18	Suspicious apps and installers of the infected OnePlus 3	43
6.19	Accessibility services table of the infected OnePlus 3	43
6.20	Bluetooth tracker detection graph after three scans, with the tracker being moved between the individual scans.	44

List of Tables

3.1	Data breaches of phone monitoring software vendors from 2015 to 2025. [12], [13]	8
5.1	Device information and the corresponding ADB commands.	18

List of Listings

- | | | |
|---|--|----|
| 1 | The YAML file with the categorisation of permissions that access sensitive information . . . | 28 |
| 2 | The YAML file of Android installer packages and their suspicious flag | 29 |

Acronyms

ADB	Android Debug Bridge
API	Application Programming Interface
APK	The file format for an Android Package
App	Software application designed to run on mobile devices
BLE	Bluetooth Low Energy
CLI	Command-line interface
DNS	Domain Name System
DV	domestic violence
GPS	Global Positioning System
GUI	Graphical User Interface
IPS	intimate partner surveillance
IPV	intimate partner violence
IT	Information Technology
Malware	Malicious computer software
OS	Operating System
OSS	Open Source Software

RSSI Received Signal Strength Indicator

Spyware Malware that secretly gathers information

SQL Structured Query Language

TFA technology-facilitated abuse

VPN Virtual private network

YAML Yet Another Markup Language

Bibliography

- [1] Sam Havron, Diana Freed, and Rahul Chatterjee, “Clinical computer security for victims of intimate partner violence,”
- [2] Rahul Chatterjee, Periwinkle Doerfler, Hadas Orgad, Sam Havron, Jackeline Palmer, Diana Freed, Karen Levy, Nicola Dell, Damon McCoy, and Thomas Ristenpart, “The spyware used in intimate partner violence,” in *2018 IEEE Symposium on Security and Privacy (SP)*, ISSN: 2375-1207, May 2018, pp. 441–458. DOI: 10.1109/SP.2018.00061. Accessed: Oct. 25, 2024. [Online]. Available: <https://ieeexplore.ieee.org/document/8418618/?arnumber=8418618>.
- [3] Maied Almansoori, Mazharul Islam, Saptarshi Ghosh, Mainack Mondal, and Rahul Chatterjee, “The web of abuse: A comprehensive analysis of online resource in the context of technology-enabled intimate partner surveillance,” in *2024 IEEE 9th European Symposium on Security and Privacy (EuroS&P)*, Vienna, Austria: IEEE, Jul. 8, 2024, pp. 773–789, ISBN: 979-8-3503-5425-6. DOI: 10.1109/EuroSP60621.2024.00048. Accessed: Sep. 22, 2024. [Online]. Available: <https://ieeexplore.ieee.org/document/10629024/>.
- [4] “Permissions on android | privacy,” Android Developers, Accessed: Mar. 8, 2025. [Online]. Available: <https://developer.android.com/guide/topics/permissions/overview>.
- [5] “Tristate location permissions,” Android Open Source Project, Accessed: Mar. 8, 2025. [Online]. Available: <https://source.android.com/docs/core/permissions/tristate-perms>.
- [6] Muawya Naser, Hussein Albazar, and Hussein Abdel-Jaber, “Mobile spyware identification and categorization: A systematic review,” *Informatica*, vol. 47, no. 8, Sep. 28, 2023, Number: 8, ISSN: 1854-3871. DOI: 10.31449/inf.v47i8.4881. Accessed: Sep. 22, 2024. [Online]. Available: <https://informatica.si/index.php/informatica/article/view/4881>.
- [7] Tin Kam Ho, “Random decision forests,” in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, vol. 1, Montreal, Que., Canada: IEEE Comput. Soc. Press, 1995,

- pp. 278–282, ISBN: 978-0-8186-7128-9. DOI: 10 . 1109 / ICDAR . 1995 . 598994. Accessed: Mar. 8, 2025. [Online]. Available: <http://ieeexplore.ieee.org/document/598994/>.
- [8] “Proximity and RSSI,” Bluetooth® Technology Website, Accessed: Apr. 1, 2025. [Online]. Available: <https://www.bluetooth.com/blog/proximity-and-rssi/>.
- [9] Diarmaid Harkin and Robert Merkel, “Technology-based responses to technology-facilitated domestic and family violence: An overview of the limits and possibilities of tech-based solutions,” *Violence Against Women*, vol. 29, no. 3, pp. 648–670, Mar. 2023, ISSN: 1077-8012, 1552-8448. DOI: 10 . 1177 / 10778012221088310. Accessed: Sep. 22, 2024. [Online]. Available: <https://journals.sagepub.com/doi/10.1177/10778012221088310>.
- [10] Renee Fiolet, Cynthia Brown, Molly Wellington, Karen Bentley, and Kelsey Hegarty, “Exploring the impact of technology-facilitated abuse and its relationship with domestic violence: A qualitative study on experts perceptions,” *Global Qualitative Nursing Research*, vol. 8, p. 2333936211028176, Jun. 29, 2021, ISSN: 2333-3936. DOI: 10 . 1177 / 23333936211028176. Accessed: Sep. 22, 2024. [Online]. Available: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8246499/>.
- [11] “FlexiSPY EXPRESS | FlexiSPY pre-installed devices,” Accessed: Mar. 10, 2025. [Online]. Available: <https://www.flexispy.com/de/express.htm>.
- [12] Joseph Cox. “Meet FlexiSpy, the company getting rich selling ‘stalkerware’ to jealous lovers,” VICE, Accessed: Mar. 10, 2025. [Online]. Available: <https://www.vice.com/en/article/meet-flexispy-the-company-getting-rich-selling-stalkerware-to-jealous-lovers/>.
- [13] “Have i been pwned: Pwned websites,” Accessed: Mar. 10, 2025. [Online]. Available: <https://haveibeenpwned.com/PwnedWebsites>.
- [14] Rose Ceccio, Sophie Stephenson, Varun Chadha, Danny Yuxing Huang, and Rahul Chatterjee, “Sneaky spy devices and defective detectors: The ecosystem of intimate partner surveillance with covert devices,”
- [15] Adeel Ehsan, Cagatay Catal, and Alok Mishra, “Detecting malware by analyzing app permissions on android platform: A systematic literature review,” *Sensors*, vol. 22, no. 20, p. 7928, Oct. 18, 2022, ISSN: 1424-8220. DOI: 10 . 3390 / s22207928. Accessed: Feb. 1, 2025. [Online]. Available: <https://www.mdpi.com/1424-8220/22/20/7928>.

- [16] Huanran Wang, Weizhe Zhang, and Hui He, “You are what the permissions told me! android malware detection based on hybrid tactics,” *Journal of Information Security and Applications*, vol. 66, p. 103 159, May 2022, ISSN: 22142126. DOI: 10 . 1016 / j . jisa . 2022 . 103159. Accessed: Feb. 1, 2025. [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S2214212622000485>.
- [17] Eslam Amer, “Permission-based approach for android malware analysis through ensemble-based voting model,” in *2021 International Mobile, Intelligent, and Ubiquitous Computing Conference (MI-UCC)*, May 2021, pp. 135–139. DOI: 10 . 1109 / MIUCC52538 . 2021 . 9447675. Accessed: Feb. 1, 2025. [Online]. Available: <https://ieeexplore.ieee.org/document/9447675/?arnumber=9447675>.
- [18] Student, Department of Applied Mathematics, Delhi Technological University, Delhi, India., Gourav Garg, Ashutosh Sharma, Student, Department of Applied Mathematics, Delhi Technological University, Delhi, India., Anshul Arora, and Assistant Professor, Department of Applied Mathematics, Delhi Technological University, Delhi, India., “SF droid android malware detection using ranked static features,” *International Journal of Recent Technology and Engineering (IJRTE)*, vol. 10, no. 1, pp. 142–152, May 3, 2021, ISSN: 22773878. DOI: 10 . 35940 / ijrte . A5804 . 0510121. Accessed: Feb. 2, 2025. [Online]. Available: <https://www.ijrte.org/portfolio-item/A58040510121/>.
- [19] Suleiman Y. Yerima and Sakir Sezer, “DroidFusion: A novel multilevel classifier fusion approach for android malware detection,” *IEEE Transactions on Cybernetics*, vol. 49, no. 2, pp. 453–466, Feb. 2019, Conference Name: IEEE Transactions on Cybernetics, ISSN: 2168-2275. DOI: 10 . 1109 / TCYB . 2017 . 2777960. Accessed: Feb. 2, 2025. [Online]. Available: <https://ieeexplore.ieee.org/document/8245867/?arnumber=8245867>.
- [20] Samaneh Mahdavifar, Andi Fitriah Abdul Kadir, Rasool Fatemi, Dima Alhadidi, and Ali A. Ghorbani, “Dynamic android malware category classification using semi-supervised deep learning,” in *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCoM/CyberSciTech)*, Aug. 2020, pp. 515–522. DOI: 10 . 1109 / DASC-PiCom-CBDCoM-CyberSciTech49142 . 2020 . 00094. Accessed: Mar. 22, 2025. [Online]. Available: <https://ieeexplore.ieee.org/document/9251198>.

- [21] Arash Habibi Lashkari, Andi Fitriah A.Kadir, Hugo Gonzalez, Kenneth Fon Mbah, and Ali A. Ghorbani, "Towards a network-based framework for android malware detection and characterization," in *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, Aug. 2017, pp. 233–23309. DOI: 10.1109/PST.2017.00035. Accessed: Mar. 22, 2025. [Online]. Available: <https://ieeexplore.ieee.org/document/8476939>.
- [22] Ajit Kumar, K.S. Kuppusamy, and G. Aghila, "FAMOUS: Forensic analysis of MOBILE devices using scoring of application permissions," *Future Generation Computer Systems*, vol. 83, pp. 158–172, Jun. 2018, ISSN: 0167739X. DOI: 10.1016/j.future.2018.02.001. Accessed: Feb. 2, 2025. [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S0167739X17323257>.
- [23] "MalwareBazaar | SHA256 a87925009ac434073430cd2d393ee4a62ca39efa28f41a1f91a20dfd460b1be3 (SpyNote)," Accessed: Jan. 20, 2025. [Online]. Available: <https://bazaar.abuse.ch/sample/a87925009ac434073430cd2d393ee4a62ca39efa28f41a1f91a20dfd460b1be3/>.
- [24] "Mobile android version market share worldwide," StatCounter Global Stats, Accessed: Apr. 7, 2025. [Online]. Available: <https://gs.statcounter.com/android-version-market-share/mobile/worldwide/>.
- [25] "Tkinter python interface to tcl/tk," Python documentation, Accessed: Mar. 18, 2025. [Online]. Available: <https://docs.python.org/3/library/tkinter.html>.
- [26] "Python UI | design GUI with python | python bindings for qt," Accessed: Mar. 18, 2025. [Online]. Available: <https://www.qt.io/qt-for-python>.
- [27] "NiceGUI," Accessed: Mar. 18, 2025. [Online]. Available: <https://nicegui.io/>.
- [28] "Mobile operating system market share worldwide," StatCounter Global Stats, Accessed: Apr. 7, 2025. [Online]. Available: <https://gs.statcounter.com/os-market-share/mobile/worldwide>.
- [29] "Configure on-device developer options | android studio | android developers," Accessed: Mar. 24, 2025. [Online]. Available: <https://developer.android.com/studio/debug/dev-options>.
- [30] "Template text parser ttp 0.9.0 documentation," Accessed: Mar. 15, 2025. [Online]. Available: <https://ttp.readthedocs.io/en/latest/index.html>.

- [31] “Android anti-reversing defenses - OWASP mobile application security,” Accessed: Mar. 24, 2025. [Online]. Available: <https://mas.owasp.org/MASTG/0x05j-Testing-Resiliency-Against-Reverse-Engineering/>.
- [32] “Delete, switch, or add users - android help,” Accessed: Mar. 17, 2025. [Online]. Available: <https://support.google.com/android/answer/2865483?hl=en>.
- [33] Enze Liu, Sumanth Rao, Sam Havron, Grant Ho, Stefan Savage, Geoffrey M. Voelker, and Damon McCoy, “No privacy among spies: Assessing the functionality and insecurity of consumer android spyware apps,” *Proceedings on Privacy Enhancing Technologies*, vol. 2023, no. 1, pp. 207–224, Jan. 2023, ISSN: 2299-0984. DOI: 10.56553/popets-2023-0013. Accessed: Sep. 22, 2024. [Online]. Available: <https://petsymposium.org/popets/2023/popets-2023-0013.php>.
- [34] “AppOpsManager | API reference,” Android Developers, Accessed: Mar. 18, 2025. [Online]. Available: <https://developer.android.com/reference/android/app/AppOpsManager>.
- [35] “About the SafetyNet attestation API deprecation | security,” Android Developers, Accessed: Mar. 17, 2025. [Online]. Available: <https://developer.android.com/privacy-and-security/safetynet/deprecation-timeline>.
- [36] Urcuqui López, Christian and Navarro, Andres, “Machine learning classifiers for android malware analysis,” in *ResearchGate*, 2016. DOI: 10.1109/ColComCon.2016.7516385. Accessed: Feb. 4, 2025. [Online]. Available: https://www.researchgate.net/publication/305649819_Machine_learning_classifiers_for_android_malware_analysis.
- [37] Alejandro Guerra-Manzanares, Hayretin Bahsi, and Sven Nömm, “KronoDroid: Time-based hybrid-featured dataset for effective android malware detection and characterization,” *Computers & Security*, vol. 110, p. 102399, Nov. 2021, ISSN: 01674048. DOI: 10.1016/j.cose.2021.102399. Accessed: Feb. 6, 2025. [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S0167404821002236>.
- [38] aleguma, *Aleguma/kronodroid*, original-date: 2021-06-11T08:09:06Z, Feb. 5, 2025. Accessed: Feb. 6, 2025. [Online]. Available: <https://github.com/aleguma/kronodroid>.
- [39] “Installing scikit-learn,” scikit-learn, Accessed: Mar. 24, 2025. [Online]. Available: <https://scikit-learn/stable/install.html>.

- [40] “Manifest.permission,” Android Developers, Accessed: Jan. 25, 2025. [Online]. Available: <https://developer.android.com/reference/android/Manifest.permission>.
- [41] Billy Lau, Jiexin Zhang, Alastair R Bereford, Daniel Thomas, and Rene Mayrhofer, “Uraniborgs device preloaded app risks scoring metrics,”
- [42] “Permissions and APIs that access sensitive information - play console help,” Accessed: Mar. 24, 2025. [Online]. Available: https://support.google.com/googleplay/android-developer/answer/9888170?visit_id=638784201005324048-366049177&rd=1.
- [43] “Hbldh/bleak: A cross platform bluetooth low energy client for python using asyncio,” Accessed: Mar. 18, 2025. [Online]. Available: <https://github.com/hbldh/bleak>.
- [44] B.G. Amidan, T.A. Ferryman, and S.K. Cooley, “Data outlier detection using the chebyshev theorem,” in *2005 IEEE Aerospace Conference*, ISSN: 1095-323X, Mar. 2005, pp. 3814–3819. DOI: 10.1109/AERO.2005.1559688. Accessed: Mar. 18, 2025. [Online]. Available: <https://ieeexplore.ieee.org/document/1559688>.
- [45] Georgia Ionescu, “Improving distance estimation in object localisation with bluetooth low energy,”
- [46] “Bluetooth-SIG / public assigned_numbers / company_identifiers / company_identifiers.yaml bitbucket,” Accessed: Mar. 18, 2025. [Online]. Available: https://bitbucket.org/bluetooth-SIG/public/src/main/assigned_numbers/company_identifiers/company_identifiers.yaml.
- [47] Array. “Libimobiledevice ù a cross-platform FOSS library written in c to communicate with iOS devices natively,” libimobiledevice, Accessed: Mar. 25, 2025. [Online]. Available: <https://libimobiledevice.org/>.
- [48] “TeamSpeak 3 - voice chat apps on google play,” Accessed: Mar. 23, 2025. [Online]. Available: https://play.google.com/store/apps/details/TeamSpeak+3+-+Voice+Chat?id=com.teamspeak.ts3client&hl=en_SG.
- [49] “Flipper zero portable multi-tool device for geeks,” Accessed: Mar. 20, 2025. [Online]. Available: <https://flipperzero.one>.
- [50] Matthew, *MatthewKuKanich/FindMyFlipper*, original-date: 2024-03-05T04:32:54Z, Mar. 19, 2025. Accessed: Mar. 20, 2025. [Online]. Available: <https://github.com/MatthewKuKanich/FindMyFlipper>.