University of Zurich[UZH]

MASTER PROJECT – Communication Systems Group, Prof. Dr. Burkhard Stiller

Design and Prototypical Implementation of HomeScout

Dylan Puser, Remo Hertig, Zhishan Yan
Zurich, Switzerland
Student ID: 14-924-054, 13-738-323, 20-752-689

Supervisor: Katharina O. E. Müller, Dr. Bruno Rodrigues,
Prof. Dr. Burkhard Stiller
Date of Submission: February 15, 2023

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Master Project
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

Bluetooth Low Energy (BLE) trackers, also known as item finders, have become a popular tool for people to keep track of their items. However, they can also be abused for nefarious purposes, for example to stalk people or to mark other people's property. While there are ways to mitigate this risk, depending on the particular tracker vendor and smartphone operating system, there is no universal application that works across operating systems and tracker vendors. To address this, the authors of this paper conceptualize HomeScout, a cross-platform mobile application with an advanced device detection and classification algorithm based on four application scenarios. They implement a proof of concept version of this application and test and validate it in a series of experiments. They find that it works well for the four defined application scenarios.

Zusammenfassung

Bluetooth Low Energy (BLE) Tracker, auch bekannt als “Item finders”, sind in den letzten Jahren zunehmend beliebter geworden, als eine Methode, die persönlichen Gegenstände zu orten. Sie werden jedoch auch für schädliche Zwecke missbraucht. So können sie zum Beispiel verwendet werden, um Leute zu stalken, oder um die Gegenstände anderer Personen zu verfolgen. Es gibt zwar Methoden, dies zu verhindern, sie sind aber abhängig vom jeweiligen Tracker oder Betriebssystem des Smartphones der Person. Um dies zu adressieren, wurde HomeScout konzipiert. Es ist eine plattformübergreifende Applikation, die mittels fortschrittlichen Detektions- und Klassifizierungsalgorithmen basierend auf vier Applikationsszenarien fungiert. Ein konzeptioneller Beweis dieser Applikation wurde implementiert und in verschiedenen Experimenten getestet und validiert. Es wurde gefunden, dass die Applikation für die Applikationsszenarien gut funktioniert.

Acknowledgments

Many thanks to Katharina O. E. Müller, Dr. Bruno Rodrigues and Prof. Dr. Burkhard Stiller for the opportunity to work on this master project at the Communication Systems Group.

We are extremely grateful to our main supervisor Katharina O. E. Müller, for her continued support and time during our weekly meetings, her knowledgeable insights and her valuable feedback throughout the duration of the project.

Contents

Chapter 1

Introduction

1.1 Motivation

With the fast-growing adoption of Internet of Things (IoT) solutions, Bluetooth Low Energy (BLE) technologies have become increasingly popular in a wide range of aspects in our daily life. According to market research, the global BLE market is expected to surpass $16.7 billion by 2026 [1]. One of the most significant and exclusive applications of BLE is asset tracking, which refers to the location-based tracking of items by attaching smart tags to them [2]. Smart tags, also known as BLE beacons, BLE trackers or item finders, are small devices that can be attached to an asset and then broadcast an Unique Identifier (UID) to allow for the identification and tracking of assets or items. An UID is an identifier that marks that device record as unique, the Media Access Control (MAC) address can be used for this purpose [3].

The main purpose of introducing these smart tags is to help users locate and monitor their belongings since the smart tags support multiple networks and communication protocols [4]. However, while the smart tags are designed to track users' own possessions, they can also be misused for malicious purposes. For example, due to the small size and ease of use, stalkers may use these smart tags to establish real-time tracking of targets by briefly accessing their personal belongings and hiding these tags within [5]. The media quickly reported articles linking smart tags to domestic abuse and stalking [6, 7]. Tracker abuse goes beyond stalking or domestic abuse, it can also be used to track valuable items such as cars [5]. On the other hand, people worry that organizations could combine and correlate the information learned from trackers to build profiles of people for nefarious purposes as more data is stored day by day. Both concerns raise important issues around personal privacy and technology surveillance, causing widespread societal concern due to the misuse of smart tags [6].

There are several BLE tracking products available on the market, such as the Apple AirTags, Tile Finders, Samsung's Galaxy SmartTags, and the Chipolo ONE Spot. While all of these smart trackers have relatively accurate location capabilities, each one has its own limitations. For example, Apple's *Find My* network prioritizes privacy, offering enhanced security and privacy protection compared to other finder networks.

It effectively shields the location of AirTag owners from both outsiders and within Apple's own ecosystem, but it only works with updated iOS devices that use the *Find My* feature [8]. The same goes for Samsung's SmartTags, which are only compatible with Galaxy phones running Android and the *SmartThings Find* app. Chipolo ONE Spot heavily relies on Apple's *Find My* network, it requires pairing up with an iOS device to locate items [9]. The Tile Finder is the only popular tracker that works with both iOS and Android, but its app can be confusing with many options and the Tile app itself cannot be removed once activated, which causes inconvenience for users who no longer use it [10].

1.2 Problem Description

It is a well-established fact that maliciously using smart tags presents a significant hazard to users in scenarios where abuse is prevalent [11]. This is due to the fact that only some smartphone systems are compatible with each other, therefore mandatory tracking protections are often needed. Consider Apple's AirTag as the case and point. Despite its release, it took the company more than six months to launch an Android app called "Tracker Detect" on December 11, 2021, in an effort to partially address this issue [12]. Despite this measure taken, the Android app has limitations as it requires frequent manual scans to locate a tracker, rendering it ineffective in practical real-life scenarios.

HomeScout (HS), a secure cross-platform application for both iOS and Android was developed to quickly detect unwanted tracking. The app enables users to scan for harmful BLE trackers using an optimized algorithm and emphasizes customizable operation, prioritizing the app interface usability for non-experts. The key contributions made include the following:

(1) Design and implementation of the open-source HomeScout application in Flutter to protect users across platforms.

(2) Conducting of tests of HomeScout and generation of an anonymized dataset for analysis.

(3) Evaluation of the algorithm's performance in regards to detecting IoT devices as dangerous or non-dangerous.

1.3 Report Outline

This Master project report is structured as follows: in chapter 2 works related to BLE scanning are discussed. In chapter 3 the background on BLE and Bluetooth Trackers is introduced. The app interface mock-ups and the architecture of the app are presented in chapter 4, along with a description of its features, highlighting the user interface design and tracking detection methods. It also includes the experimental setup for the evaluation. In chapter 5 the implementation of the app is discussed. In chapter 6, the data gathered from testing is analyzed. The limitations of the current design are discussed in chapter 7, followed by suggestions for future work and final conclusions.

Chapter 2

Related Work

With the rise of IoT and the ubiquity of BLE devices, a lot of research has gone into the tracking and position-finding of devices using BLE. [13] create an indoor positioning system (IPS) using BLE beacons' received signal strength indicator (RSSI) to approximate the location of a device moving through the location. [14] develop a smartphone based application, called LocBLE, that allows users to assess the location of nearby BLE beacons in a fine-grained manner.

Following the rise in popularity of BLE trackers, especially with the release of Apple's AirTags in 2021, some research has shifted to address benefits and issues concerning these trackers. [15] show that BLE devices can be used to track patients with dementia. As more elaborate trackers, such as ones using GPS, would need to be regularly removed for charging or for cleaning of the patients, BLE devices are uniquely suited for this purpose. Similarly, [16] propose and demonstrate a system that uses BLE and UWB within a location to detect emergency situations of elderly or infirm people.

A lot of other works occupy themselves with the privacy concerns raised by BLE devices. [17] analyse wearable fitness trackers, which periodically synchronise data with the user's smartphone. They show that due to a majority of fitness trackers using unchanged BLE addresses, they can be abused to track and learn information from the user wearing them. Of particular concern is the transmission of a user's gait, which, as it is fairly unique for each person, allows for a sort of fingerprinting of a user. [18] showed how BLE devices can also be fingerprinted at the physical level and show mitigation strategies to limit this practice.

Many past studies have examined various aspects of Apple's wireless services in particular. Analysis by [19] proposes and assesses a proof-of-concept protocol that utilizes Apple's *Find My* network and a custom tag with a basic microcontroller. [20, 21] conduct a reverse engineering of the Apple *Find My* system and allow a set of BLE devices to appear as tags inside Apple's *Find My* network. Both analyses demonstrate how the Apple *Find My* service can serve as a privacy-sensitive tool for crowd monitoring and how it may unintentionally reveal a person's location, if deliberately tracked. Another study by [22] explores the *Handoff* services, which allow seamless communication between multiple Apple devices under the same iCloud account with enabled *Find My* network.

They have been found to potentially undermine random MAC addresses and to identify devices belonging to a single user. Several privacy issues are also uncovered, including device fingerprinting and activity tracking. These works support the need to examine proprietary systems that could impact billions of users worldwide. [23] show that Apple's safety alerts, which are intended to warn a user that they are being tracked, can be circumvented by creating a custom BLE tracker that can participate in the *Find My* network, while not triggering the safety alerts.

Some of the research concerns itself with solving some of the issues illustrated above. [24] propose and demonstrate an enhanced *Find My* protocol, *Blind My*, that introduces additional cryptographic verification into the protocol to guarantee that devices can be detected when tracking. [25] introduce *PrivateFind*, a finder protocol that improves on privacy concerns of existing protocols by improving on parts of the entire crowd search system. [26] too, introduce a design for a secure and privacy-retaining crowdsourced tracking system, called SEC_{ROW}.

[12] on the other hand take a different approach, by creating a smartphone application that lets users more accurately or more actively detect if they are being tracked. In this vein they introduce *AirGuard*, an Android application that addresses issues of Apple's own Android app, *Tracker Detect*. [27] too present an Android application that improves on the speed and precision of tracker detection.

Chapter 3

Background

3.1 Bluetooth Low Energy

Bluetooth Low Energy (BLE) is short-range wireless technology similar to classic Bluetooth. While classic Bluetooth has achieved wide-spread adoption, it is not the ideal technology for every use case. Especially for use cases related to the Internet of Things (IoT), which are characterized by asymmetric communication (e.g. a smart sensor transmits more data and only occasionally receives data) and low power consumption (e.g battery powered sensors that last for years). For these use cases, classic Bluetooth is too complex, expensive and uses too much power. As a response, *Bluetooth SIG*, the designer of classic Bluetooth, developed a new Bluetooth variant, "Bluetooth Low Energy", specifically for low power consumption, low complexity and low cost applications.

It uses 40 physical channels (2 MHz wide) in the 2.4 GHz band [28]. At the physical layer there exist two types of channels, advertising channels (3) and general purpose channels (37). The advertising channels play a central role in BLE, they are used for devices to broadcast packets to other nearby devices, without having to establish a connection first.

BLE is organized as a layered stack of protocols and modules. They are grouped into two parts, *host*, and *controller*. Usually, but not necessarily, the *controller* is implemented directly in a System-on-Chip (SoC), whereas the *host* is implemented in software and provided as a driver by the operating system. This organization is illustrated in Figure 3.1. The complete BLE stack is of considerable complexity and the details of each layer are out of scope of this work, with the exception of the Generic Access Profile (GAP) and the link layer [28]. In the remainder of this section, only those parts of BLE are discussed, which are of relevance for this work's use case, namely scanning for common *Bluetooth trackers* such as the Apple AirTag. Some of the details are discussed exemplarily for the AirTag, because their use of BLE has been studied considerably in literature, among the available *Bluetooth trackers* [29, 12, 30].

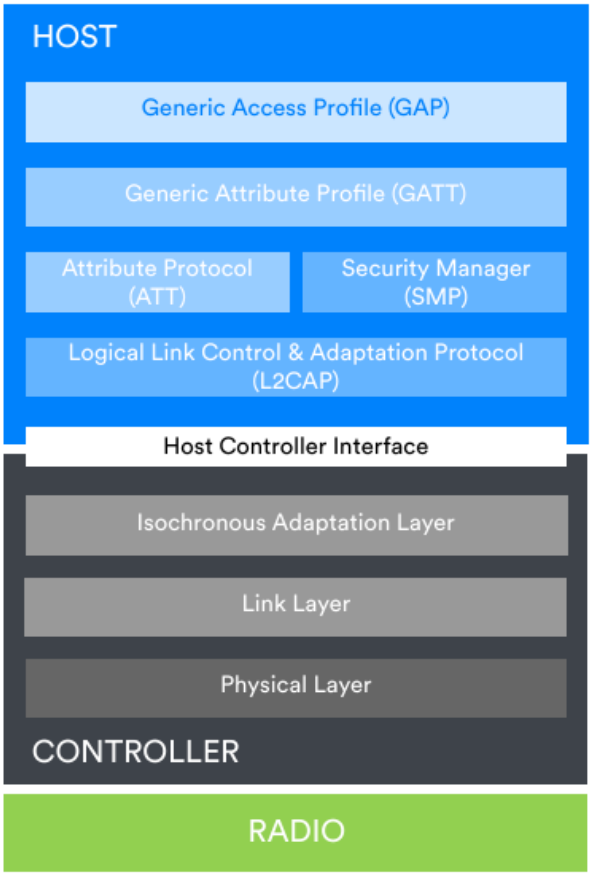


Figure 3.1: BLE stack

Source: [31]

3.1.1 Generic Access Profile (GAP)

GAP is concerned with device discovery and connection. For this, it defines procedures for *advertising packets*, which allows for connectionless communication. The device which intends to send data connectionless is said to operate in the *Broadcaster role*. In this role, the host can instruct the controller via the *Host Controller Interface* to enable advertising, which puts the *Link Layer* in the *Advertising state*. In this state, the *Link Layer* sends packets regularly during *Advertising events*. A particular advertising packet can be transmitted up to three times, because it's copied on all three primary advertising channels. Which increases the probability of successful transmission if there are other interfering devices or radio-frequency sources [28].

Conversely, a device that intends to search for this advertising signal is said to operate in the *Observer role*. In this role, the host, respectively the user application, instructs the BLE controller via the *Host Controller Interface* to enable scanning, which puts the *Link Layer* in the *Scanning* state. In this state, the *Link Layer* listens for advertising packets on the primary advertising channels and forwards the content back to the host upon receipt [28].

3.1.2 Link Layer

The *Link Layer* in BLE is concerned with the low-level packets and the management of links. The actual transmission of bits over the air on a particular channel is handled by the *Physical Layer* (see Figure 3.1). These two layers are analogous to the first two *ISO Reference model* layers [32].

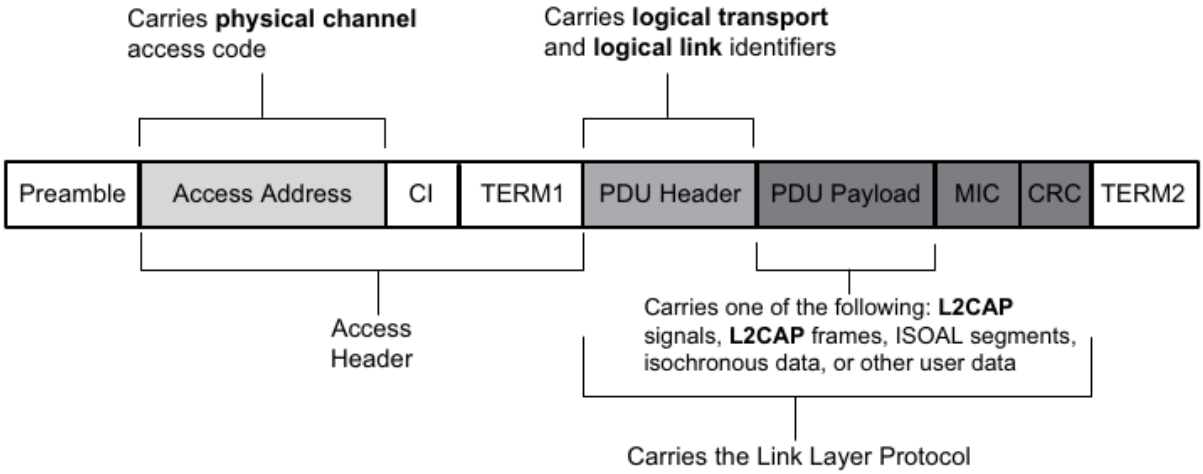


Figure 3.2: BLE packet structure

Source: [28]

In BLE a packet consists of a Preamble, an Access-Address, Protocol Data Unit (PDU) and a Cyclic redundancy check (CRC) (see Figure 3.2). In the *Advertising state* the *Link Layer* populates the packet with an *Advertising physical channel PDU* consisting of a 16-bit header and a payload. The header contains the length of the payload and the type of the PDU [28].

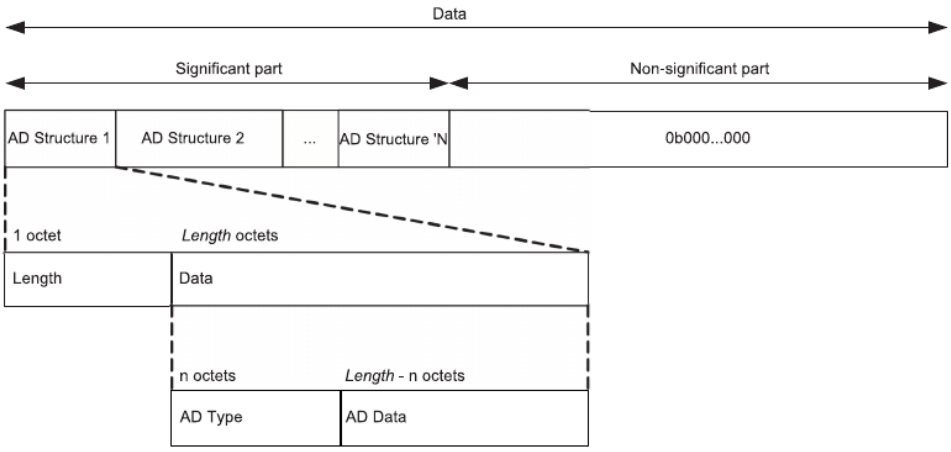


Figure 3.3: Advertising data format

Source: [28]

An Apple AirTag will use the *ADV_IND* PDU type in its advertising packets [29]. The payload of type *ADV_IND* contains a 6 bytes address part and a 32 byte data part. The data part can consist of multiple sub fields, each prefixed by a byte indicating its length as shown in Figure 3.3. Each sub field contains one byte indicating the *AD Type* and a *AD Data* part, which can take up all the remaining space in the *PDU Payload*. The maximum size for a data sub field in the *ADV_IND* payload is thus 30 bytes[1] [28].

3.2 Bluetooth Trackers

Bluetooth trackers are devices that allow for localization of objects using the Bluetooth LE technology, which can include indoor positioning. They have a wide range of applications, among others in Industrial IoT, as an example for asset tracking in a warehouse. Common consumer applications include the recovery of lost items such as keys or wallets. In this context, Bluetooth trackers are sometimes referred to as ”key finders”, *Bluetooth finder* [25] or *Item Finders* [30].

A Bluetooth tracker is usually a small device consisting of a battery, antenna and a Bluetooth chip. To keep the costs of such a tracker low, they don't include additional capabilities such as mobile internet access or a Global Positioning System (GPS) receiver, which would also require a much larger battery. This device is physically attached to the object of interest. Whoever is looking for that object needs a second computing device with Bluetooth capabilities and a special application. While the object of interest is within range of the finder device, they can occasionally share information using Bluetooth. To locate the object, the finder device can utilize information from the Bluetooth exchange to estimate the object's location. It can also instruct the Bluetooth tracker to play a sound to assist the user with locating it. One such method is to use the received signal strength indicator (RSSI) of the Bluetooth connection as an indicator for relative distance [33, 34]. This method uses physical properties of electro-magnetic information transmission. Particularly, the inverse-square law of electromagnetic radiation, which states that the power of a point source is inversely proportional to the squared distance. However, in practice, RSSI based ranging is more complex. The orientation of the finder's device can influence the measured signal strength (depending on the used antenna). Additionally, the Bluetooth protocol itself optimizes for power consumption and can reduce the transmission power as long as the transmission error rates are acceptable [33].

3.2.1 Crowd-Sourced Location Tracking

When the object of interest is no longer within range to establish a Bluetooth connection, the finder's application can use historical information of the location of last contact to indicate where the object might be. However, the object might no longer be at the last seen location, for example because it was lost somewhere else after the last contact point or because the object with the attached tracker was stolen.

[1] 32 bytes - 1 byte Length - 1 byte *AD Type*

In such a case, the Bluetooth tracker purely on its own cannot be localized anymore, because the Bluetooth tracker can only communicate to devices within a limited radius. To remedy this situation, the signal would have to be relayed to the finder by an additional third device. An innovative approach for this problem was commercially first introduced by the startup *Tile* in 2013, where they use crowd sourcing, *crowd search* or *crowd finding* [35]. Large hardware companies, such as Apple and Samsung, have added similar functionalities to their platforms a few years later.

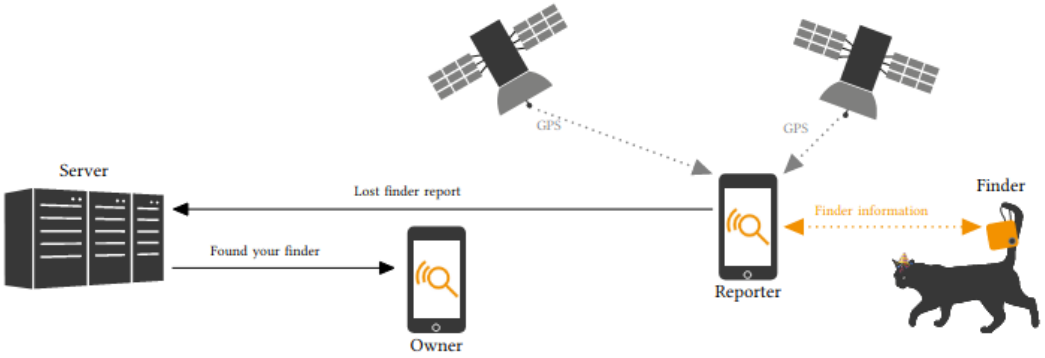


Figure 3.4: Illustration of crowd-sourced location tracking

Source: [25]

The general idea in *Crowd-Sourced Location Tracking* is that the signal of a lost Bluetooth tracker could be picked up by many other networking-capable devices, such as the ubiquitous smartphones, which usually have Bluetooth capability. The lost Bluetooth tracker needs to transmit two pieces of information to the third-party finder: its status, that the tracker is lost and an unique identifier such that the owner of the device can identify the Bluetooth tracker. The third-party finder then needs to relay that information to the owner of that device together with the precise location of where the signal was picked up. With this information the finder device of the owner can inform the user that the lost Bluetooth tracker, hopefully with the attached object, has been found at an approximate location [25]. This mechanism is visualized in Figure 3.4. If a large enough crowd equipped with smartphones and a specific crowd-finder application were to continuously listen for signals from lost Bluetooth trackers, then they could be found reliably. Important for this functionality is thus the size of the crowd or the finder network.

3.2.2 Tile Trackers

Network The main crowd-source network used by Tile trackers constitutes of users of the Tile app on Apple and Android devices [35]. Tile does not publicly report the size of its crowd sourcing network. Tile trackers also use the crowd-source network of other companies, such as the *Amazon Sidewalk* network, which is used by the *Amazon Echo* products [36].

3.2.3 Apple AirTag Trackers

AirTag is a Bluetooth tracker released in 2021 by Apple [37]. It contains a nRF52832 System-on-Chip (SoC) which supports BLE 5.2 and Near-field communication (NFC). Additionally it has an Apple U1 chip for Ultra-Wideband (UWB) communication [29]. AirTags use Apple's *Find My* network which uses *iPhones* and *iPads* with GPS capabilities [20]. According to [29], AirTags use undirected BLE advertising to inform nearby devices of its presence. For this it uses the *Manufacturer Specific Data* field in the *ADV_IND* PDU by setting the *AD Type* to 0xFF [38]. The *Manufacturer Specific Data* field is at least 2 bytes long and uses the first two bytes to identify the manufacturer [39]. In case of the AirTag, the manufacturer is 0x004C (Apple, Inc.) [38]. The rest of the *Manufacturer Specific Data* is in a proprietary and used for multiple of Apple's products. This proprietary format starts with an *Apple payload type* field, followed by an *Apple payload length* field. An *Apple payload type* of 0x12 indicates that the message is related to Apple's *Find My* network [29, 20].

3.2.4 Samsung Galaxy Trackers

Samsung announced their Bluetooth tracker products *Galaxy SmartTag* and *Galaxy SmartTag+* in 2021. The device is equipped with both BLE and UWB technology, allowing it to accurately locate missing items [40]. The *Galaxy SmartTag+* has strong detection abilities with the help of *SmartThings Find* app, which allows users to locate lost items on a map using BLE and the *Galaxy device* network. To use the feature, the user needs to let their *Galaxy* smartphone or other compatible device with the *Galaxy SmartTag* app login to assist in finding lost SmartTags for others. The other advantage of using *Galaxy SmartTag+* is allowing users to find lost items using *AR Finder* thanks to UWB technology. Once a missing item is found, the *Galaxy SmartTag+* will guide the user with a user-friendly interface showing the distance to the *Galaxy SmartTag+* and by pointing the user in its direction [41].

Network Samsung reported in 2022 that their crowd-sourcing network comprises of over 200 million devices [42]. Both Samsung and Tile took a similar approach to networking their devices and application [43, 44]. Therefore these networks operate very similarly: Devices and items use BLE to broadcast their presence to nearby supported devices, which then identify and upload their location to a cloud service [45].

3.2.5 Chipolo One Spot

Chipolo One Spot is a key tracking tag that helps people find personal items with the Apple *Find My* network, which is made up of hundreds of millions of Apple devices around the world [9]. As Chipolo relies on the entire Apple *Find My* network, it works exclusively with the *Find My* app and is not shareable. Therefore other users can never see the location of the user's Chipolo One Spot [46].

3.3 Apple Ecosystem

3.3.1 Offline Finding

The Offline Finding (OF) system was introduced by Apple in 2019. It is a crowd-sourced location tracking system for offline devices and now officially is called the *Find My* network [47]. The basic idea behind OF is to allow users to locate their lost or stolen devices even when they are in a location where they do not have any internet access.

To use the feature successfully, users need to sign in to their iCloud account and have the *Find My* settings enabled. Once enabled, the so-called *finder* device can detect the presence of the offline device to periodically send an approximate location back to iCloud [19]. iCloud is an online cloud-based service for all Apple services that handle online data storage and synchronization via Apples' servers. All apple users will have their own account that synchronizes itself via iCloud, and OF uses iCloud to share keys across all owner devices that are signed in to the same Apple account [48].

Following this, the owners can use other Apple devices to log in to iCloud to check the location of their lost item. Additionally, users can also set their lost device to "Lost Mode", which allows them to remotely display a message on the screen, or to lock or erase the device if needed [49].

3.3.2 Tracking Workflow

Apple's *Find My* network tracks devices that opt in via anonymous crowd-sourced location reports. Once the devices are marked as lost, its owner can access location reports through their iCloud account and view them using the *Find My* application [50]. These smart tags are constantly transmitting, even without the presence of their owners. To set it up, the iCloud and tag owner create a public-private key pair using API calls for tracking the device [51]. The process of recovering lost devices involves the following:

1. **Broadcast:** *Find My* accessories are initialized with a private-public key pair that allows devices to generate an infinite number of rotating key pairs that help link devices. Once the device loses its Bluetooth or internet connection to the owner's device, it will broadcast the Bluetooth advertisements with rotating public keys [20].
2. **Upload reports:** Finder devices that detect the Bluetooth advertisements extract the public key, and then create a temporary private-public key pair to encrypt the finder's location report, containing (1) the beacon reception time, (2) the confidence about that contact, (3) the MAC address, (4) the key location information, and (5) an authentication label to further validate the report. These reports are securely uploaded along with the finder's public key to Apple's iCloud after a certain time period. In addition to these data fields, the batch of reports from a single finder is annotated with a timestamp of when it was received on the server side [19].
3. **Download reports:** With the public keys, the owner can download and retrieve available location reports on the lost device with secure HTTPS GET requests.

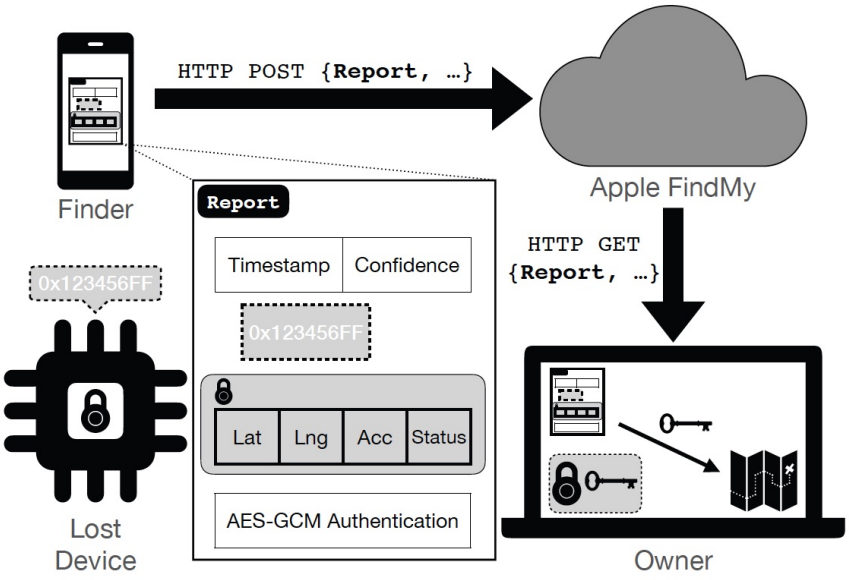


Figure 3.5: Delay in sensing and reporting a tag

3.3.3 Item Safety Alerts

With the launch of AirTags, Apple also introduced Item Safety Alert (ISA) into iOS, with the intent to notify the owner of an iPhone that they may be tracked by a malicious tracking device. Since Apple's *Find My* network has significantly more participating devices than other similar services, it makes it a highly effective tool for finding lost items at relatively low cost. The item safety alerts on iPhone appear when the same lost message is received consistently, exceeding certain elapsed time and distance thresholds [52]. If a device keeps receiving the same lost message, it infers that there is an unknown AirTag tracking device nearby and alerts the users [24]. The Item Safety Alert (ISA) includes a warning of a possible malicious tracking device and instructions on how to identify and disable it.

3.3.4 Find My Network Accessory Program

The old *Find My* app made it easy to locate Apple devices while protecting user privacy, but the new updated *Find My Network Accessory Program* extends these capabilities by locating missing devices even if there is no internet connection, as the *Find My* network is a crowd-sourced network of Apple devices that use BLE technology to detect missing items nearby, and then to report their approximate location back to the owner.

The updated *Find My* app was introduced by Apple in April 2021. the goal of the app is to allow third-party products to use the private finding capabilities of Apple's *Find My* network, which includes hundreds of millions of Apple devices. Due to the *Find My Network Accessory Program*, the vast and global *Find My* network opens up to third-party device manufacturers to build products utilizing the service, so their customers can use the *Find My* app to keep track of their important devices, as well as keep up with friends and family [47].

3.4 Other Tracking Detection Applications

3.4.1 Tracker Detect

Tracker Detect[2] is an Android app developed by Apple in December 2021. This app was Apple's answer to numerous concerns about malicious tracking of Android users by AirTags [53].

However, *Tracker Detect* only offers limited features with certain restrictions applied. For example, it only allows users to scan manually for nearby *Find My* accessories. This can potentially reduce the likelihood of detections [19]. Once an accessory is found, the user has to wait 10 minutes before they can play a sound on the device to locate it. Unlike its counterpart, it also provides limited functionality to scan for Apple AirTags while the app is open, but not in the background [51].

3.4.2 AirGuard

AirGuard is an Android app by [12], developed to address some of the shortcomings of *Tracker Detect* and to protect Android users from malicious uses of AirTags. It includes automatic background BLE scanning, similar to the one on iOS, rather than having to manually initiate scans as with *Tracker Detect*. AirGuard performs such a scan every 15 minutes. It filters found devices for *Find My* advertisements in the *separated* BLE state and stores them along with the device's current location in a local database. It then runs a tracker classification algorithm, in which it iterates through all devices in the database and checks (1) if the device's advertisements have been received for longer than 30 minutes, (2) if the device has been picked up a minimum of three times, (3) if the device has followed the user for more than 400 meters, and (4) if the user has not been issued a tracking alert by AirGuard in the last seven hours. If all four conditions apply, the user is alerted.

Upon receiving a tracking alert, the user can view more details on the tracking device. It includes a map with the locations where the device was detected, and allows for the user to play a sound on the tracker, ignore it and more.

AirGuard also includes a manual scan mode. It can be used to manually get the approximate distance, using RSSI as a basis for approximation, of nearby devices. Lastly, the app contains a dashboard that provides a user with an overview of how many trackers were found, when the last one was detected, as well as a risk level card. The risk level card is green if there were no tracking devices detected in the past two weeks, orange if more than one device was found to be following a user for less than 24 hours, or red, if more than one device was detected as to be following a user for more than 24 hours. This system provides users with a simple insight into their risk of being tracked, without having to evaluate tracking activity themselves.

[2] https://play.google.com/store/apps/details?id=com.apple.trackerdetect

Chapter 4

Design

4.1 Requirements

In a first phase of the Master project, several requirements were extracted from the task description of the Master project [54]. These requirements were categorized into design and implementation constraints (4.1.1), functional (4.1.2) or nonfunctional requirements (4.1.3). In addition to the requirements stated in the task description, we made further assumptions about the environment and usage of the application which are described in 4.1.4. Based on these requirements several use-cases were modeled as shown in Figure 4.1.

4.1.1 Design and Implementation Constraints

Design and Implementation constraints are issues that restrict the available options for development, including but not limited to hardware availability and limitations or language requirements [55].

- HS must operate on Android mobile phones supporting BLE and at least with version Android 12.
- HS must run on handheld devices with limited memory, storage, and computing capacity.
- The user interface of HS must adapt to small screen sizes.
- HS must be able to run in the background, whereas it is subject to constraints from the Android OS scheduler (battery saving).
- HS must be implemented in the Dart language using the Flutter framework.

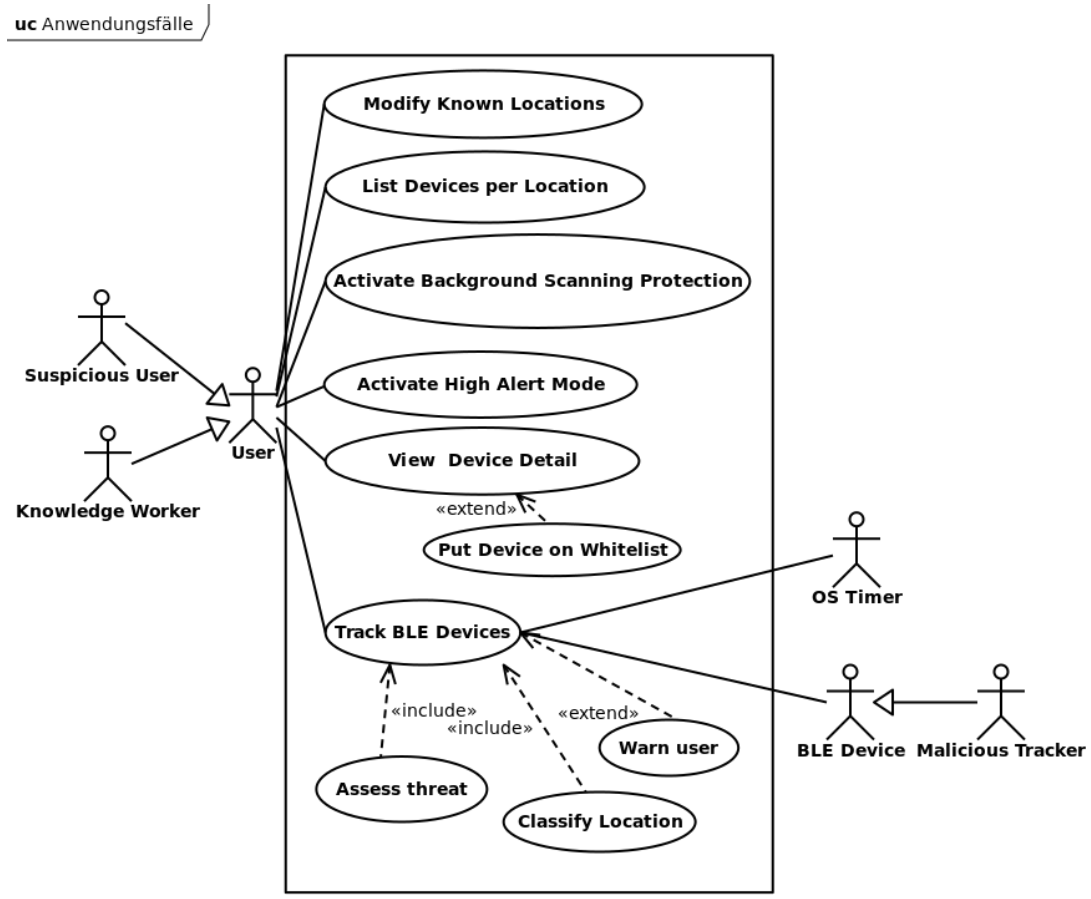


Figure 4.1: Modelled Use-cases

.

4.1.2 Functional Requirements

Functional requirements describe which functions can be performed by the application [55].

1. The systems should access precise location data (Lat/Long) from the respective OS API.
2. The system should scan nearby BLE devices.
3. The system should persist information about nearby BLE devices across app restarts.
4. The system should classify discovered devices as either dangerous or not.
5. The system should notify the user if it is close to a dangerous device (blocklisted).
6. The user can mark a device as safe. All such marked devices are visible on an allowlist.
7. The system should classify the current location according to the user-defined location zones.

4.1.3 Nonfunctional Requirements

Nonfunctional requirements describe properties of the applications, for example, quality attributes of the software [55].

Extensibility It should be easy to extend HS with new functionality and data sources.

Efficiency HS should use system resources sparingly to not drain the smartphone's battery. During passive tracking, it should use up no more than 10% of the battery.

Storage Capacity The persistent data of HS should use up no more than 1Gb of flash storage.

Portability HS primarily runs on Android devices but shall be usable on iOS devices with minimal modifications.

Privacy The collected geolocations are personal data. HS should only keep data as long as necessary to perform its functions and remove older data regularly.

4.1.4 Additional Design Assumptions

The design space of potential solutions spanned by the original task description was still of considerable size. To keep the scope of the Master project reasonable, it was further constrained by making two additional assumptions about the operating environment.

Focus on Dedicated Bluetooth Trackers Bluetooth trackers as described in section 3.2 are dedicated devices to track objects. But they are not the only devices that can be used for tracking purposes. For example, other devices from Apple such as Macbooks are also connected to Apple's *Find My* network and supposedly emit similar BLE signals as an AirTag when they lose connection to the internet and the owner's phone. Technically, it should be possible to track objects or even people using laptops or other similar devices from Apple. However, these scenarios were deemed as out of scope for the current project. Thus, the main focus will be on the tracker detection of the provided Bluetooth trackers, which can be identified by their BLE signals. The application will however be designed to allow for the detection of generic BLE devices.

Focus on Outdoor Tracking Relatively accurate location measurements are necessary to detect if a malicious device is following a user. In an outdoor setting the location can be measured on standard smartphones with reasonable accuracy of about 5 meters by using only GPS [56]. When using GPS signals in combination with other signals as it is commonly performed in modern mobile operation systems, the accuracy can be further improved. However, tracking the movement of a smartphone in an indoor setting is much more difficult. While it is technically possible to receive GPS signals even indoors, the indoor accuracy in concrete buildings varies with the size of the building as well as the composition of the surrounding buildings [57].

For smaller buildings, it is usually below 10 meters but can worsen up to 60 meters for large multi-story buildings. Another problem indoors is that the signal acquisition takes much longer and thus also increases the energy consumption of the smartphone [58].

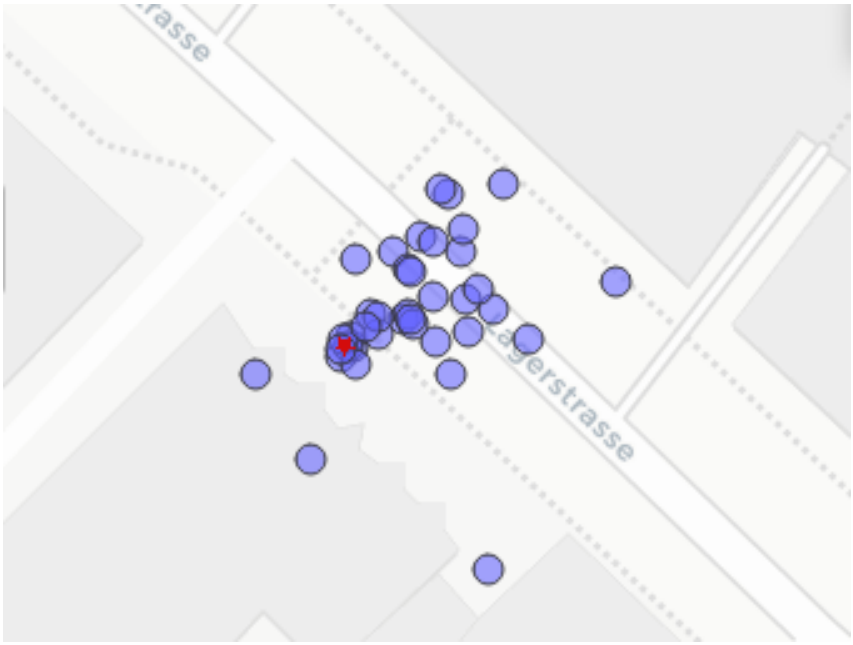


Figure 4.2: Visualization of location drift

.

The problem with GPS accuracy when not in ideal open field conditions was observed when periodically recording the location as reported by an Android smartphone. In Figure 4.2, the holder of the smartphone was mostly stationary at the point indicated in red and recorded the locations in regular intervals. However the location measurements varied noticeably.

Given the difficulties of accurate indoor localization, the scope of the project is restricted to outdoor scenarios. Therefore, the assumption is made that the typical user of the HomeScout application will only be using it to detect trackers over larger outdoor distances and not within buildings.

4.2 Application Scenarios

One goal of HS is to help users with tracker detection in a multitude of different scenarios. Users should have control over things such as the aggressiveness of tracker detection, detection based on location and more. For this purpose, four scenarios were defined. In the following subsections, each will be elaborated upon further.

4.2.1 Normal Mode

The first scenario concerns a user who suspects they are being tracked. They can then open the app and enable scanning. Unless constrained by the specific device, this can be done both with the app open and active, or with the app in the background. The app will then periodically scan for nearby devices, filter and annotate them, as well as gather additional data such as the user's location. It then, based on data on the device, the device's detection history, the user's location and some other factors, evaluates the threat level of the detected trackers and potentially alerts the user.

4.2.2 Heightened Awareness Mode

Heightened awareness mode is a mode of the application that users can enable or disable. It is intended for cases where a user is more suspicious of being tracked, e.g. if they left their bag unattended for a while, or for cases where a user might be at a higher risk of being tracked, e.g. late at night walking home alone. In such cases, the user can enable heightened awareness mode. This mode both changes the scan frequency and duration, as well as the conditions in the threat classification algorithm that lead to a device being assessed as a threat.

4.2.3 Location Designation

Often it is desirable to mark certain locations as "safe", i.e. finding a tracker in this location should not be of concern. This could the case in an office building, where there are likely to be a multitude of trackers belonging to other people. In such a situation, one would receive too many false positives to be helpful. Originally, the concept was for the threat assessment to ignore devices in such cases. This was however more elegantly solved by requiring both the user and a tracker to move with each other for a certain distance and time, eliminating cases where one would be stationary for a long time. Therefore, locations were reworked to alter the conditions that lead to the threat assessment to classify a tracker as a threat. In the case of a "safe" location, it will increase the conditions needed for a tracker to be deemed a threat.

Alternatively, one might want to mark certain areas as particularly "dangerous", for example a bad neighbourhood. In such locations then, as opposed to the "safe" locations, the threat assessment should be more sensitive and be quicker to alert users.

For HS, a rudimentary proof of concept was developed, that let users mark their current location, as well as a radius, as either safe or unsafe, as well as see what location they are currently in.

It is worth noting, that "safe" in this context means safe from being tracked, not safe in a more general sense. For example, someone might want to designate their home as "dangerous", as being tracked to your home would be of high risk.

4.2.4 Device Block- and Allowlisting

An important part of a tracker detection application is the ability to mark detected trackers. Then, that information can be taken into account on a repeat detection. Two scenarios take place:

First, the detected tracker is familiar and benign to the user. This would for example be the case if the tracker belonged to a friend of the user. In such a situation, the user could mark the device as "harmless", or in more technical terms, the user could add it to the allowlist. This would then be taken into account during the threat classification on a repeat detection.

Second, the detected tracker is unknown and there exists reasonable suspicion that it might be dangerous. In that case, the user could mark it as "dangerous", or again more technically, add it to the blocklist. This will lead to the device being more quickly classified as a threat if it is detected again.

4.3 Application Architecture

4.3.1 Logical View

Figure 4.3 shows a high-level view into the logical components of the application and the flow of information. Two data sources, the BLE scanner and the GPS location service, provide data to a location based classifier. It, together with data from the allowlist and blocklist, are used by the device risk assessor.

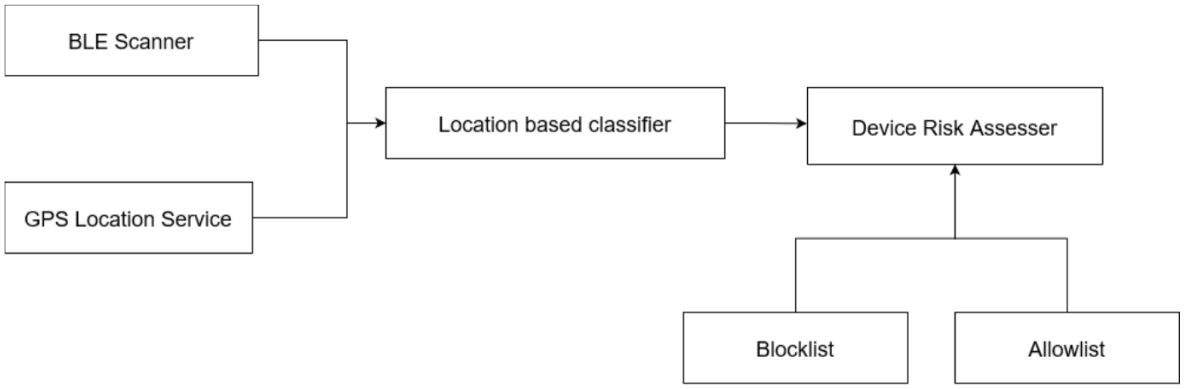


Figure 4.3: Information flow

.

4.3.2 Data View

The application uses a SQLite database to store the data received by scanning for BLE devices. The used database scheme is shown in Figure 4.4. The core functionality of the application uses only four tables. For the experiments an additional table is used to log certain events used for data analysis.

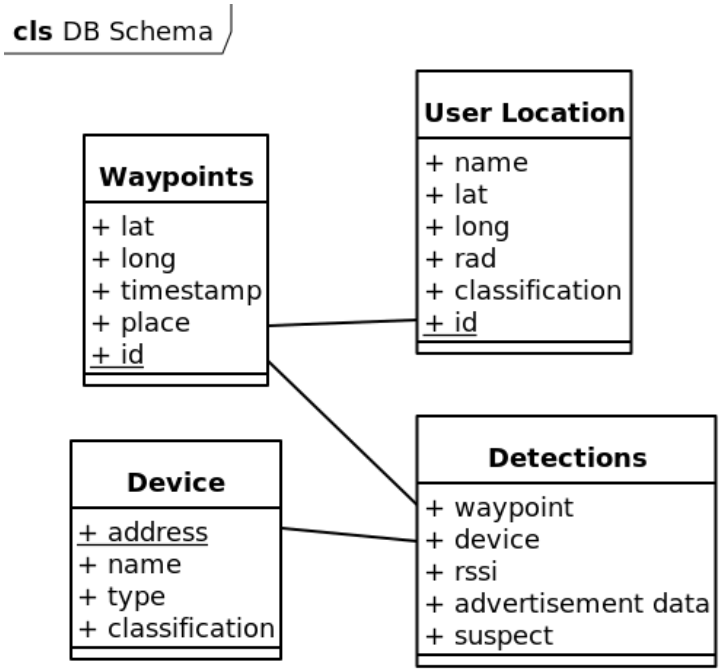


Figure 4.4: Database schema

.

4.3.3 Development Structure

The application was developed using a three-layer architecture, similar to the one described by [59]. A schematic view can be seen in Figure 4.5. The bottom-most layer is the data layer, where the business logic resides. It in turn is comprised of repositories and data sources. Data sources, as the name suggests, are the points of origin of data. In our application, two examples would be the database and the BLE scanner. Repositories then are used to expose the data to the rest of the app. Each repository is responsible for for a certain type of data, e.g. a *device* repository in our case. The next layer, the domain layer, is optional and encapsulates additional business logic that might be used across multiple repositories. HomeScout, as an example, makes use of a *device* domain. Lastly, the UI layer is responsible for displaying the data on screen. It is mainly comprised of widgets, the name given to components built with Flutter.

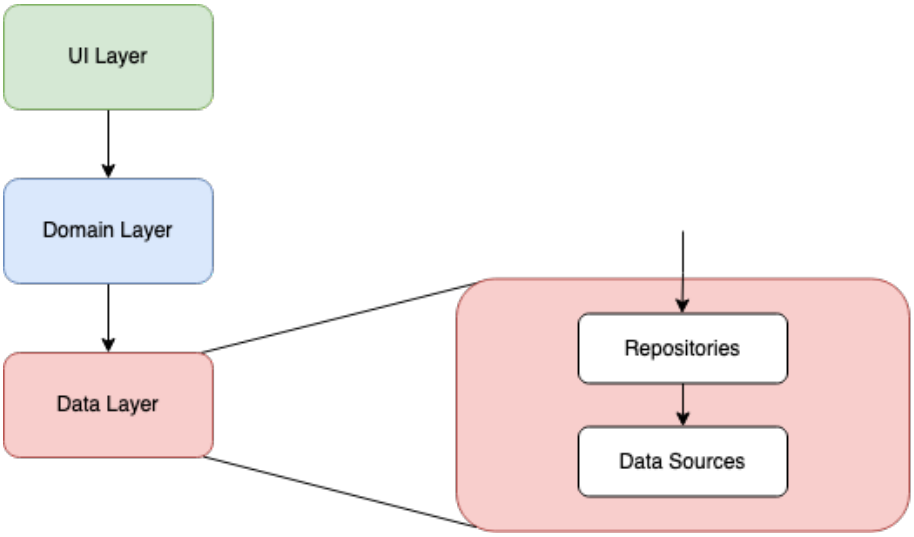


Figure 4.5: The architecture used to develop the app. It shows the 3 layers and a zoomed-in look at the data layer.

.

4.3.4 Threat Service

The threat service is responsible for taking a scanned device and assessing its threat based on certain factors. These factors include the device's journey, the user's journey since it first spotted the device, the saved user locations a user might currently be in and whether or not the device has previously been allow- or blocklisted. As the heightened awareness mode scenario requires the app to be able to switch between two modes of differing threat assessment strategies, and to keep the app open to future extension, the *strategy design pattern* was used [60]. It delegates the logic to a separate and exchangeable interface, which can be implemented to create a new strategy. Figure 4.6 shows a diagram of this service.

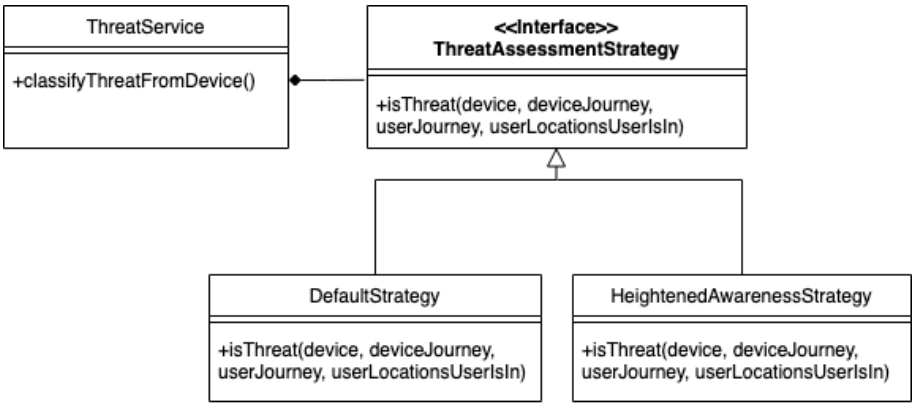


Figure 4.6: Class diagram of the threat service, which uses the strategy pattern to implement different threat assessment strategies.

Furthermore, to simplify the setting of strategies, the factory pattern was used. Thus, the strategy can easily be set using an enum. Listing 1 exemplifies this.

```
set threatAssessmentStrategy(ThreatAssessmentStrategies threatAssessmentStrategy) {
    _threatAssessmentStrategy = ThreatAssessmentStrategyFactory
                                        .of(threatAssessmentStrategy);
}
```

Listing 1: Example listing of the threat assessment strategy being set.

4.4 Tracker Detection

The tracker detection contains three classifiers. A classifier for the location to determine whether the current location is marked as extra dangerous or save. A classifier to detect the device type which uses the data published by each device during their BLE advertisements. Finally a classifier to determine whether a device is potentially malicious. The interconnections between these subsystems are shown in Figure 4.7.

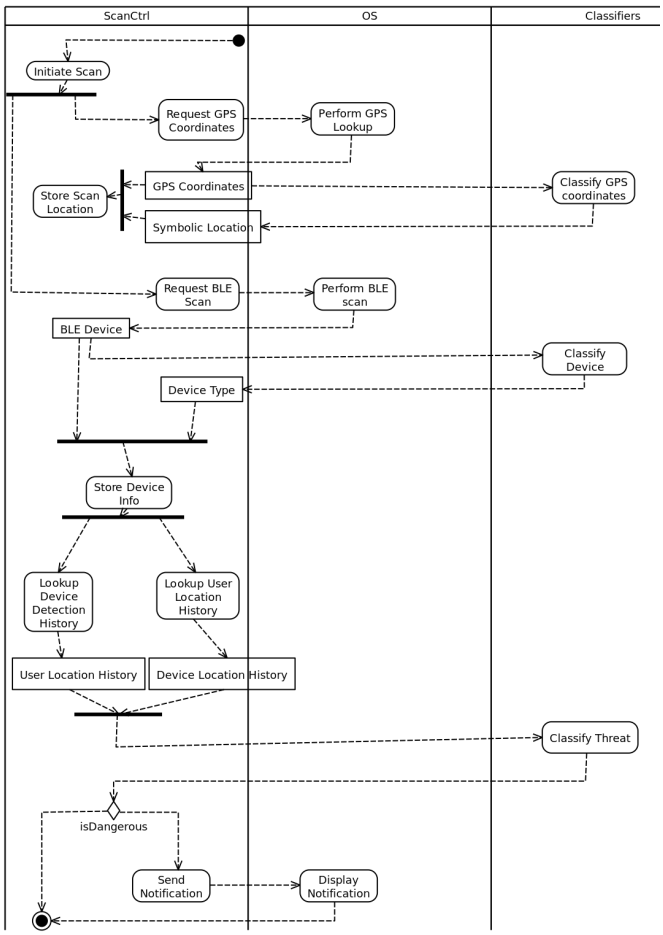


Figure 4.7: Scanner Overview

.

4.4.1 Detection Algorithm

Normal Mode

Normal mode is the default mode the app runs in when scanning is enabled. It initiates periodic scans of nearby BLE devices and the user's current location. Both location and devices are classified and stored in a local database. Then, using data already gathered on a device, such as its previous detections, the threat is assessed. The exact algorithm used to classify a device as a threat can be seen in Figure 4.8.

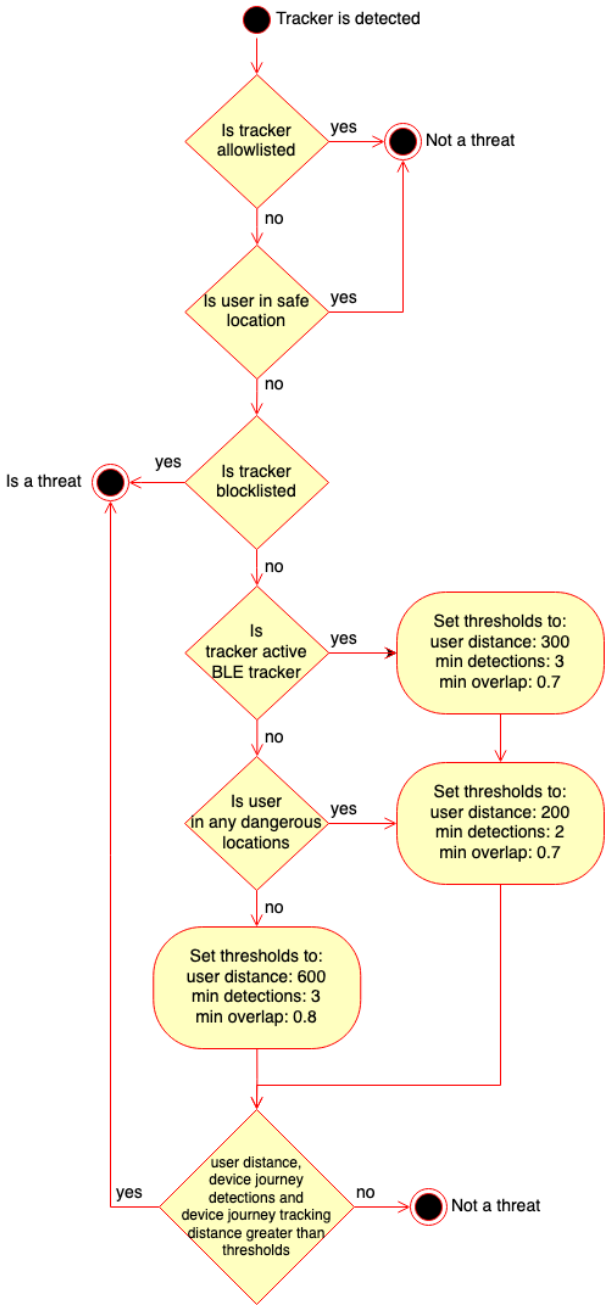


Figure 4.8: The algorithm of the normal mode threat classification.

.

Heightened Awareness Mode

Heightened awareness mode, as described in section 4.2.2, is a application mode that users can enable if they more strongly suspect they are being tracked. It modifies two aspects of HS. It modifies two aspects of HS. One the one hand, it modifies the frequency and duration of scans. The frequency is changed from scanning every 15 minutes in normal mode to every minute in heightened awareness mode. The duration of a scan is increased from 4 seconds in normal mode to 5 seconds. On the other hand, heightened awareness mode also changes the threat classification algorithm. It is shown in Figure 4.9.

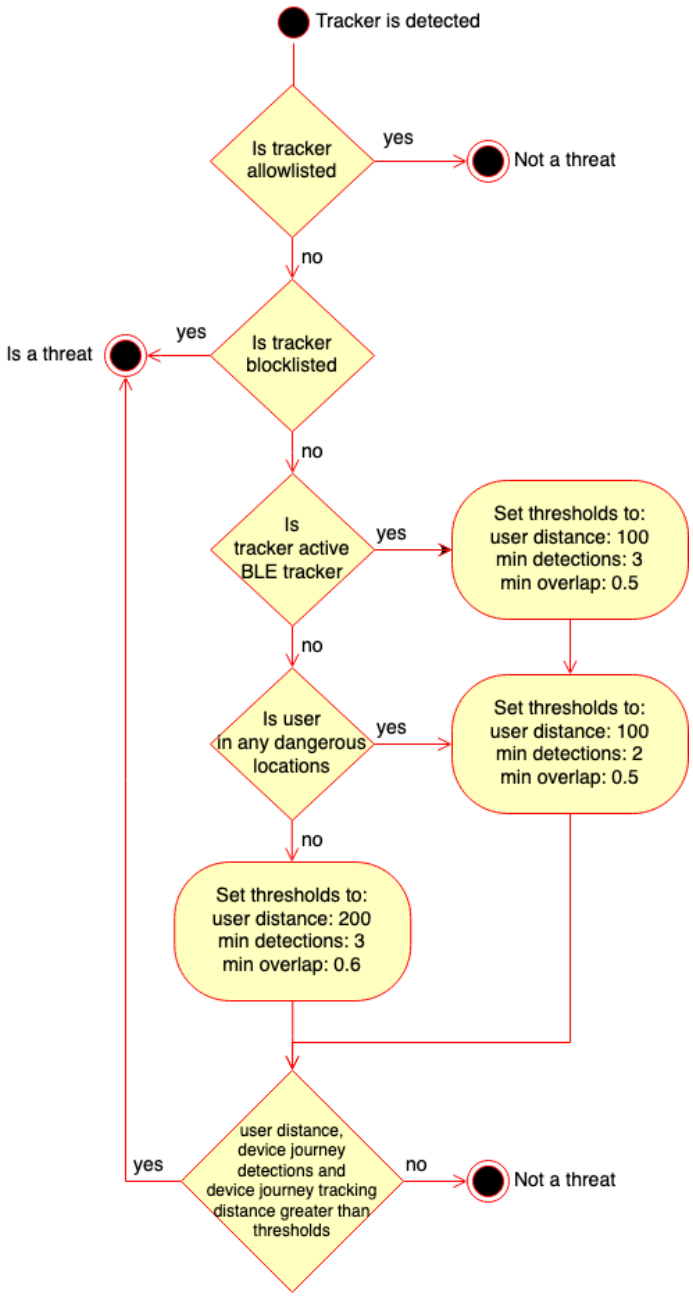


Figure 4.9: The algorithm of the heightened awareness mode threat classification.

.

4.5 User Interface

The following detailed designs showcase the progression throughout the project, where the application was designed and iterated on from the initial paper-based wireframes prototype to a medium-high fidelity prototype made with Adobe XD and finally to a fully developed proof-of-concept app made with Flutter. From the iterations of the app design, it is apparent that the goal was to design and develop a product that was functional and user-friendly to the application users. A screenshot of the final homepage is shown in Figure 4.10.

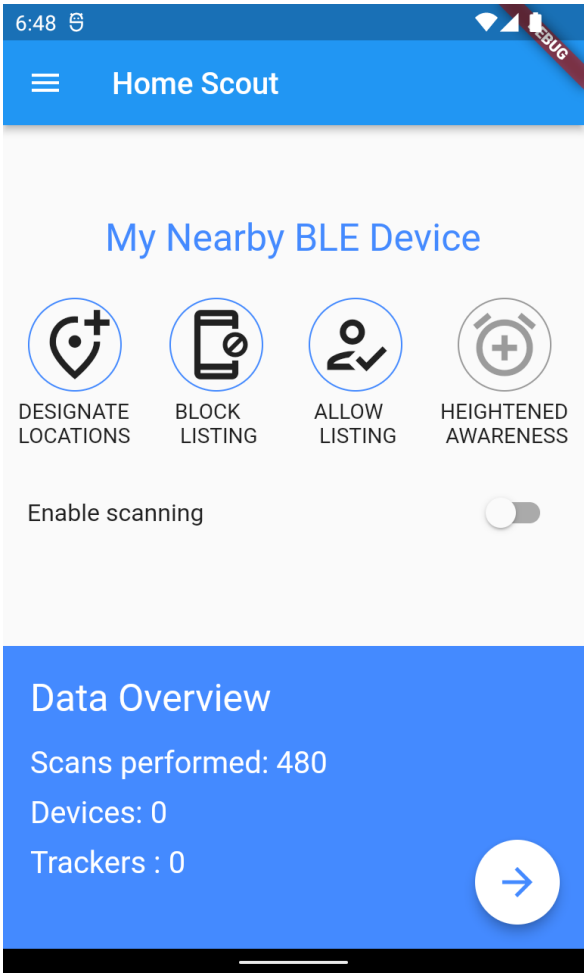
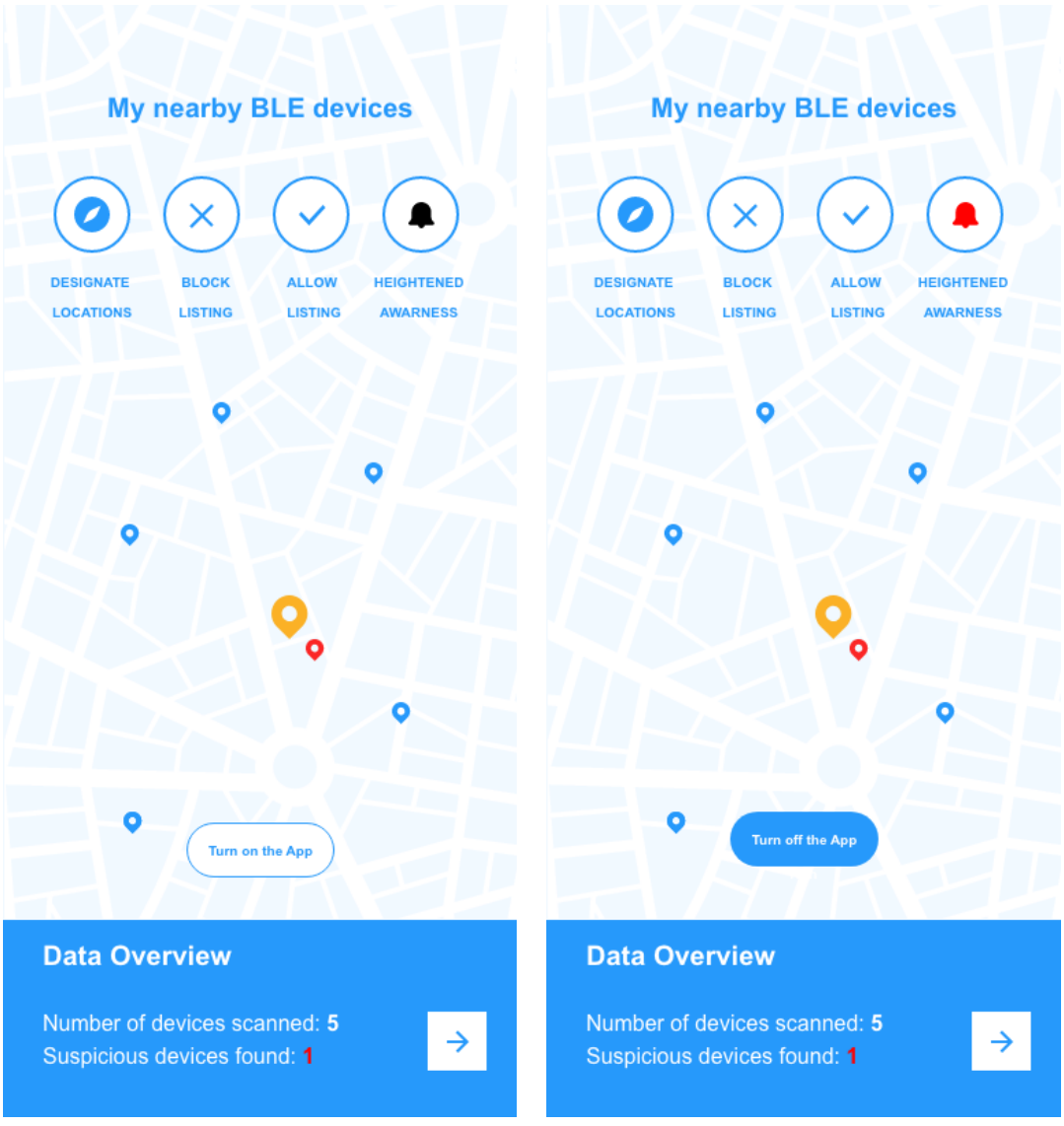


Figure 4.10: Homepage of HomeScout App (Screenshot)

4.5.1 Homepage

The first page that initializes and refreshes the application would be the homepage, where users check the real-time map to assure their safety, a mockup is shown in Figure 4.11. Besides the location-display in the background, there is a navigation bar that consists of four main options which allow users to access various screens easily.

In the middle of the screen, there is a control button that gives the option for users to turn the current heightened awareness mode on or off, and the icon on the top-right hand that represents the current mode will also switch colors accordingly to further inform and bring awareness to users. At the bottom of the page, the total number of scanned devices and the number of suspicious devices found will be displayed. To further check on the scanning statistics, users can click the support button on the right.



(a) Normal Mode (b) Heightened Awareness Mode

Figure 4.11: Homepage of the HomeScout App (Mockup)

4.5.2 Designate Locations

The location designate page, shown in Figure 4.12, allows users to save the current location with a certain label. There are two labels to choose from, the safe and the dangerous label. Underneath the label selection, the radius range can be adjusted by users. The idea of having labels and distance settings for each location is that the app will take the safety (from being tracked) of a location into account when assessing the threat of a device. Below that, a list of locations the user has created and their corresponding data are shown. Locations that a user is currently in, receive a green icon instead of the blue one. A location can be deleted with a button to the right of each location.

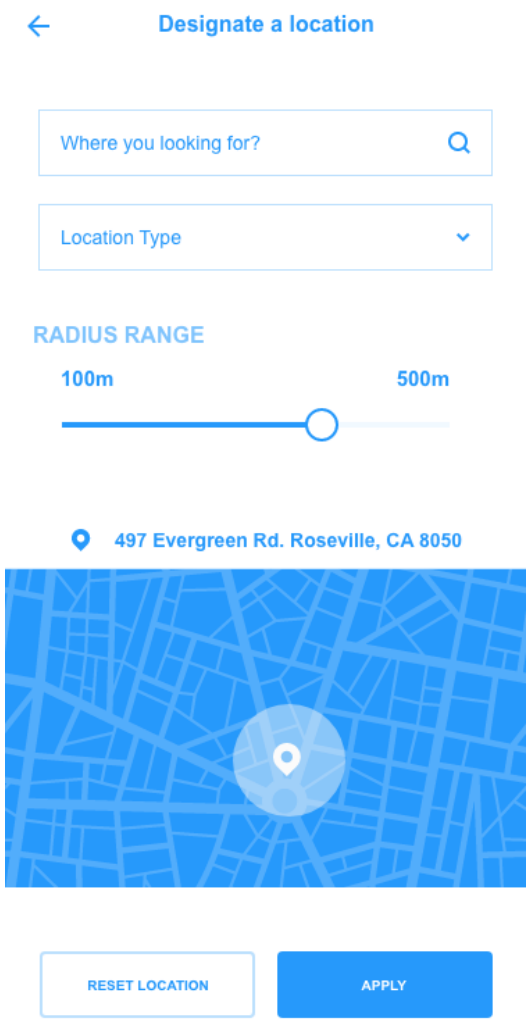
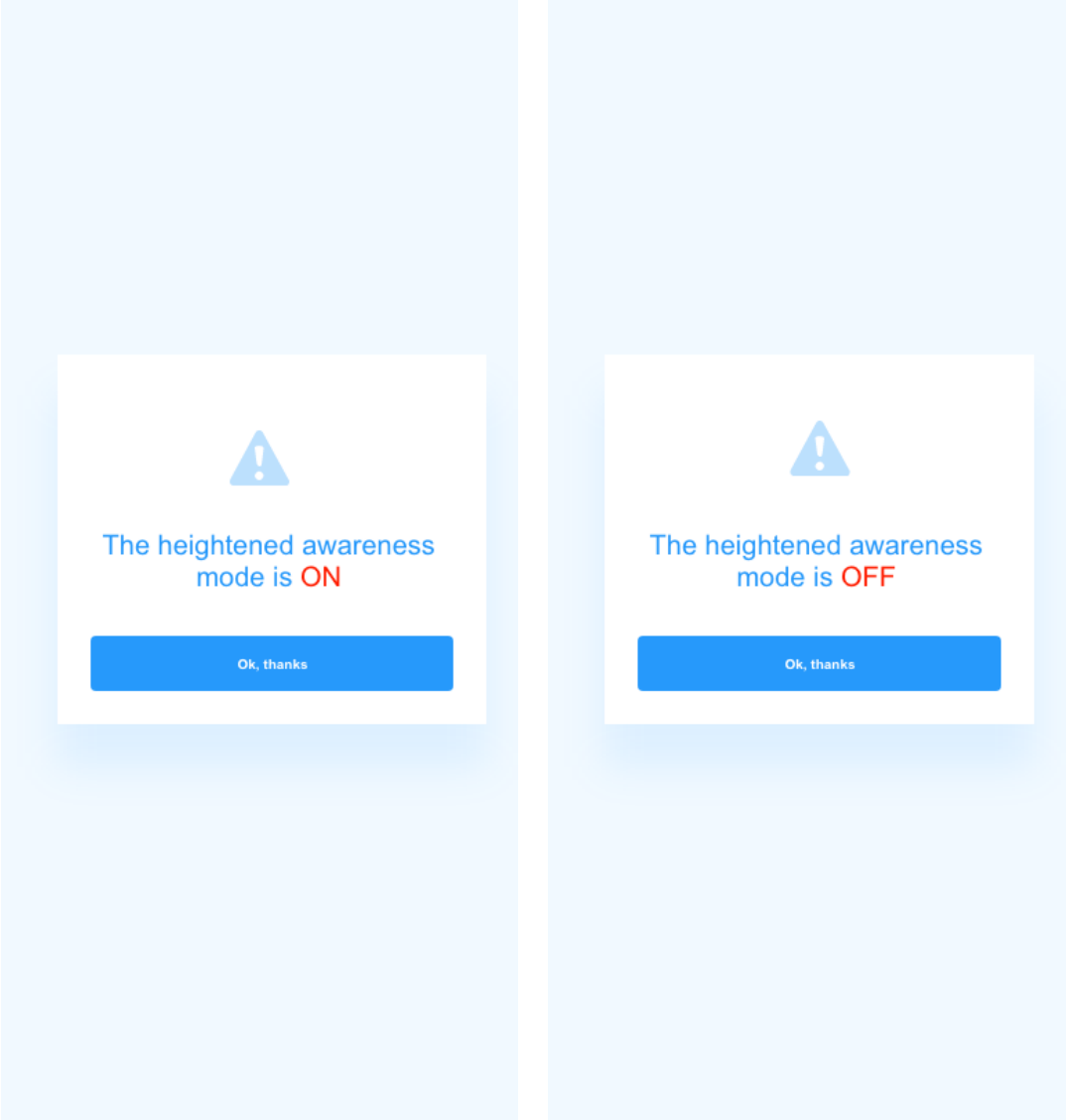


Figure 4.12: Location Designate page of HomeScout App (Mockup)

4.5.3 Pop-out Window

The pop-out window, shown in Figure 4.13, appears a few seconds after the key features have been turned on or turned off, drawing the user's attention to the current operating state of the selected feature.

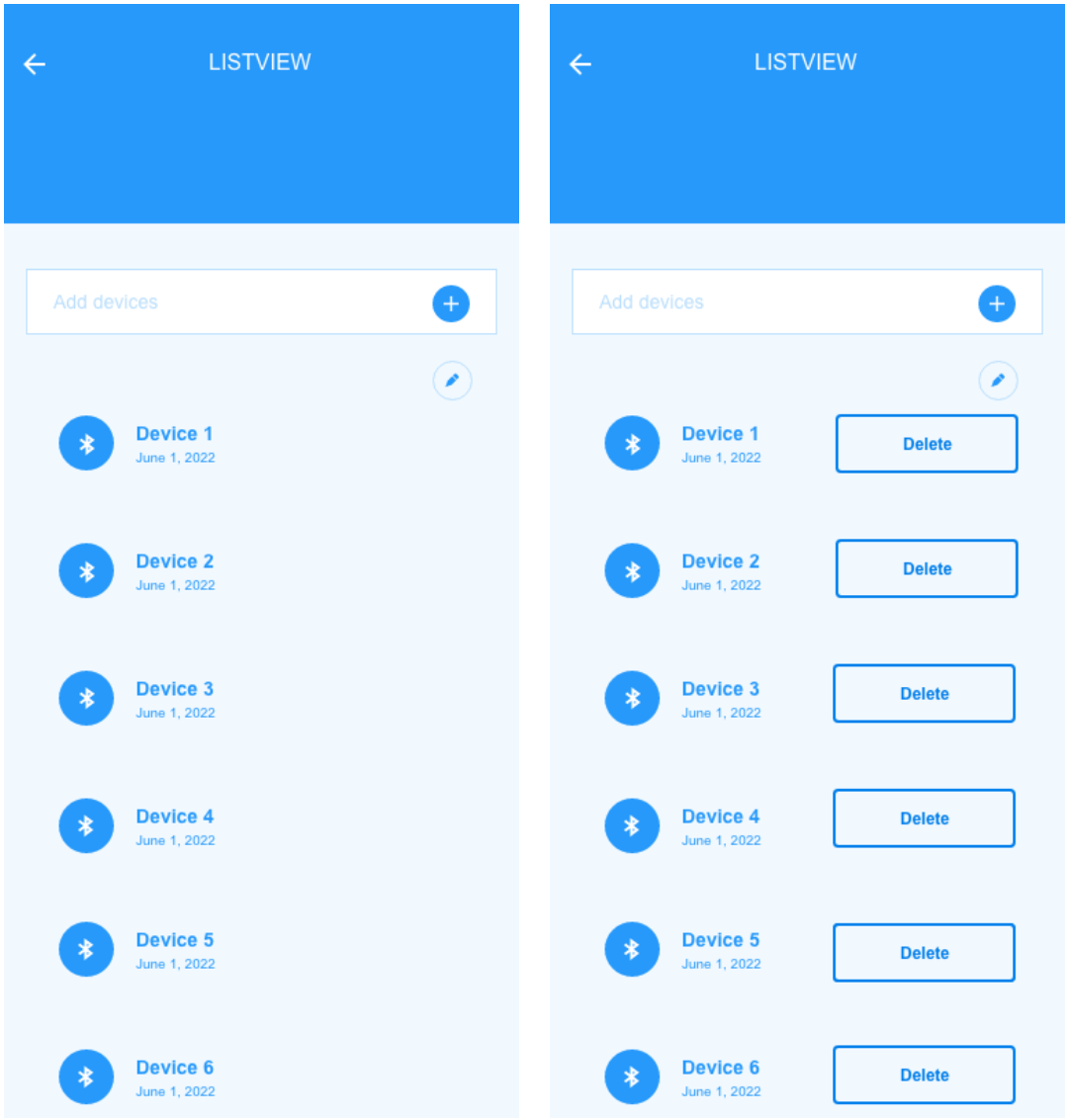


(a) Heightened Awareness Mode (b) Normal Mode

Figure 4.13: Pop-out Window example of HomeScout App (Mockup)

4.5.4 Lists

There are two lists in our App, one is for collecting all the devices that are considered to be “friendly” and other list is a collection of “malicious” devices. Both lists have the same design, which consists of a search bar on the top that allows the user to search the devices. More information can be viewed by pressing a device. By clicking the *Edit* icon on the upper right screen, devices can be deleted. The two lists are shown in Figure 4.14.



(a) Allowlist (b) Blocklist

Figure 4.14: Allow- and blocklist pages of HomeScout App (Mockup)

4.5.5 Device Details

This is an information page that is designed for users to view more details on the selected device. Followed by the tracking map, the details of the device consist of *tracking device*, *first time seen with you*, *total tracking time*, and *total tracking distance*. This data will bring awareness to the users on a detected tracker, and allows a user to add it to the *allowlist* or *blocklist*. A mockup of the device details page is shown in Figure 4.15.

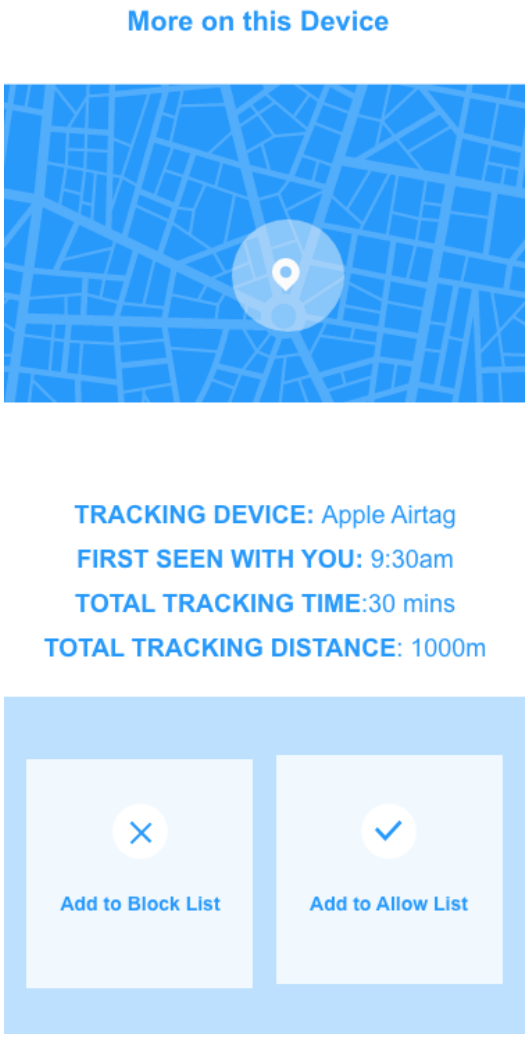


Figure 4.15: Device details page of HomeScout App (Mockup)

4.5.6 Tracking Alert

The tracking alert page will only be triggered if a threatening tracker is detected. It is shown in Figure 4.16. Tracking information such as device name and the starting time will also be displayed. This page works as a warning for users to get informed about the ongoing tracking and then to further take safety precautions.

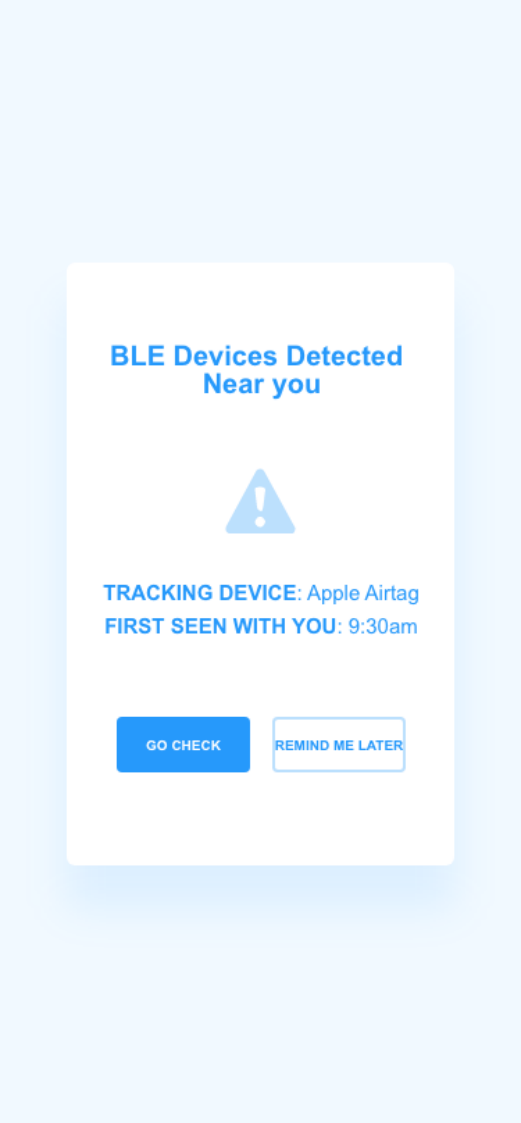


Figure 4.16: Tracking alert page of HomeScout App (Mockup)

4.6 Experimental Design

The performance of the HS application regarding the four application scenarios will be assessed with real-world experiments. Using the platform specific knowledge obtained during the application implementation, several experiments were devised. These experiments cover all four application scenarios.

The target deployment of the HS application is with regular end-users living under various different conditions. The app should work not just in a controlled lab environment, but also in a busy city bus. To account for these circumstances, it was opted to evaluate the app in a non-lab environment with varying potential interference sources. For this, the diverse set of trackers made available to us were carried throughout different locations, for example by going grocery shopping with the trackers in the bag, real-world usage was closely mimicked.

To gather diverse data samples the data collection was distributed onto all team members. To collect a data sample, each team member loaded the HS application with the experiment extension onto the phone used in the experiment and walked outside, depending on the experiment conditions, with or without trackers. For better reproducibility, the exact instructions for the experimental setup can be found in section A.2. The experiment app-extension stores additional metadata in the database. After the data collection step, the SQLite database was extracted from the phone's internal storage using the Flutter development tools respectively Xcode. The experiment logs were then extracted from the database and preprocessed using an R script. For the analysis all the extracted samples from multiple phones were pooled into one dataset.

Experiment Questions

The four application scenarios are evaluated using the following questions.

1. Does the Heightened Awareness Mode detect trackers faster than the normal mode?
2. How does the false alarm rate differ for the Heightened Awareness Mode and the baseline mode?
3. Do blocklisted devices lead to quicker alerts? Do allowlisted devices not trigger alerts?
4. Do locations designated as safe reduce the alarm rate? Do dangerous locations increase it?

Measures

For most experiments the following measures were recorded:

1. Duration
2. Distance traveled
3. Number of trackers carried
4. Number of BLE scans performed
5. Number of trackers detected
6. Time to first tracker detection
7. Distance to first tracker detection

4.6.1 Experiment 1: Tracker Detection

The goal of this experiment is to assess the performance and detection rates of the HS application when known trackers are in close proximity. In this experiment, a user carries one or more trackers in their pocket or bag and travels with the HS application active. The data for this experiment is collected with the following procedure:

Conditions

1. User puts one or mulitple active trackers in a pocket or bag
2. User loads the HS application

Experiment Round

1. User notes the start time and conditions
2. User activates the experiment mode
3. User activates the HS scanner, either normal or Heightened Awareness Mode
4. User travels around for at least xm and $y$min
5. User acknowledges any *tracker detection* notifications
6. User stops experiment mode

Measures For this experiment the following measures were recorded:

1. Duration
2. Distance traveled
3. Number of trackers carried
4. Number of BLE scans performed
5. Number of trackers detected
6. Time to first tracker detection
7. Distance to first tracker detection

Detection rate

The detection rate will be estimated with multiple approaches. First, as the ratio of experiments runs that lead to a tracker notification and those that do not. However, this measure will be dependent on the experiment run time and distance traveled. Since it is planned to collect data in different circumstances like a walk in a park or a ride in a tram, the duration and distances will vary greatly across experiments. This measure will be called `detection_rate_experiment`.

Second, as the rate of trackers detected within some reasonable time interval. According to the used tracker detection algorithm, the minimum number of scans in which a device has to show up in the *Heightened Awareness Mode* is 3. For this mode, a detection statistic could be the ratio of trackers detected at the third scan. This statistic has to be interpreted carefully, since not every scan is able to pick up the signal of a particular tracker, due to interference or the tracker not advertising during the scan period. This measure will be called `detection_rate_fast`. Alternatively, the numbers of scans can be interpreted as a time interval, since the scan period is constant. Then another detection statistic could be the ratio of trackers detected within 15 minutes, given that there was sufficient movement that would allow such a detection in the first place. This measure will be called `detection_rate_t15m`.

4.6.2 Experiment 2: Allowlisting

The goal of this experiment is to check whether a specific device can be added to the allowlist manually by the user to avoid receiving threat alerts. This experiment will be performed based on the previous setting and the result of the experiment “Tracker detection”. The data for this experiment was collected as described on the following page.

Conditions

1. The active trackers can be accurately labeled
2. User puts one or multiple active trackers in a pocket or bag
3. User loads the HS application

Experiment Round 1

1. User notes conditions
2. User activates the experiment mode
3. User activates the HS scanner, either normal or Heightened Awareness Mode
4. User travels around to get tracker detection notifications
5. User adds the tracker to the allowlist
6. User stops experiment mode

Experiment Round 2

1. User notes conditions
2. User activates the experiment mode
3. User activates the HS scanner
4. User travels around to get tracker detection notifications
5. User finds no alerts are shown for devices that were added to the allow list
6. User stops experiment mode

Experiment Round 3

1. User selects allow page from the homepage
2. User clicks the ‘edit’ icon from the listing page
3. User deletes the device from the listing page
4. User notes conditions
5. User activates the experiment mode
6. User activates the HS scanner
7. User travels around to get tracker detection notifications
8. User finds that alerts are again shown for devices that were deleted from the allow list

Measures For this experiment the following measures were recorded:

1. Duration
2. Distance traveled
3. Number of trackers detected added to the allow list
4. User finds no alerts for the device that has been added to the allow list
5. User finds alerts for the device that has been deleted from the allow list

4.6.3 Experiment 3: Blocklisting

The goal of this experiment is to check whether a specific device can be added to the blocklist manually by the user to immediately be notified of it on a repeat detection. This experiment will be performed based on the previous setting and the result of the experiment "Tracker detection". The data for this experiment is collected with the following procedure:

Conditions

1. The active trackers can be accurately labeled
2. User puts one or multiple active trackers in a pocket or bag
3. User loads the HS application

Experiment Round 1

1. User notes conditions
2. User activates the experiment mode
3. User activates the HS scanner, either normal or Heightened Awareness Mode
4. User travels around to get tracker detection notifications
5. User adds the tracker to the blocklist
6. User stops experiment mode

Experiment Round 2

1. User notes conditions
2. User activates the experiment mode
3. User activates the HS scanner
4. User travels around to get tracker detection notifications
5. User very quickly is notified of previously blocklisted device having been detected
6. User stops experiment mode

Measures For this experiment the following measures were recorded:

1. Duration
2. Distance traveled
3. Number of trackers detected added to the block list
4. User finds quick notification of previously blocklisted device having been detected

4.6.4 Experiment 4: False alarms

The goal of this experiment is to estimate the false alarm rate of the HS application when *no* known trackers are in proximity. In this experiment a user carries **no** trackers in a pocket or bag and travels with the HS application active. The data for this experiment is collected with the same procedure as in section 4.6.1, but under different conditions:

Conditions

1. User does **not** carry any trackers
2. User loads the HS application

Measures For this experiment the following measures were recorded:

1. Duration
2. Distance traveled
3. Number of BLE scans performed
4. Number of trackers detected

4.6.5 Experiment 5: Location Designation

The goal of this experiment is to assess the performance of the geofencing component and to check whether the user is able to designate certain locations as safe or dangerous. The data for this experiment is collected with the following procedure:

Conditions

1. The active trackers can be accurately labeled
2. User puts one or multiple active trackers in a pocket or bag
3. User loads the HS application

Experiment Round 1

1. User notes conditions
2. User determines a large area that they will remain in for the duration of the experiment
3. User activates the experiment mode
4. User activates the HS scanner, either normal or Heightened Awareness Mode
5. User travels around while remaining in the area to get tracker detection notifications
6. User stops experiment mode

Experiment Round 2

1. User notes conditions
2. User designates the location they previously chose and remained in as safe or dangerous in the app
3. User activates the experiment mode
4. User activates the HomeScout scanner
5. User travels around while remaining in the area for approximately the same amount of time or trackers detected as in the previous experiment
6. User receives less or more alerts respectively than in the previous experiment
7. User stops experiment mode

Measures For this experiment the following measures were recorded:

1. Number of trackers detected during experiment round 1
2. Time to first tracker detection during experiment round 1
3. Number of trackers detected during experiment round 2
4. Time to first tracker detection during experiment round 2

Chapter 5

Implementation

5.1 Dependencies

Flutter The HomeScout is implemented using Google's Flutter framework. The latest available version at the start of the project was chosen, which was *Flutter 3.0.3*. Google releases a new Flutter version approximately every 3 months, where new features are added and some functionalities are deprecated and removed [61]. Initially, dependency versions were kept fixed, to not introduce complications through upgrades. At the end of August 2022, a new version was released: *Flutter 3.3.0*, which provided some additional useful functionalities regarding the user interface (UI), specifically the extended *Material Design 3* support. Unfortunately, some features used in the user interface (UI) code was already deprecated, thus a simple upgrade was not possible and would have required a rewrite of some of the UI code.

However, later on in the development process it became tedious to not use the latest version of Flutter due to Google's peculiar documentation policies: only the documentation of the latest Flutter version is published and after each release the old documentation is deleted from the website. After a while, the documentation became less helpful, because it was also was not marked if a particular function or parameter was only recently introduced, which slowed down development a bit. Therefore, in October the UI components were rewritten to allow for an upgrade to *Flutter 3.3.6*. On the plus side, this enabled the use of some newer *Material Design 3* components on the main screen to facilitate the creation of the labeled buttons with icons.

GPS Flutter itself does not provide any APIs for accessing GPS on smartphones. However, they publish recommendations called the *Flutter Favorite Program*[1] for third-party packages of high quality for certain use cases. The *geolocator* plugin[2] is the recommended way to access the location APIs on Android and iOS.

[1] https://docs.flutter.dev/development/packages-and-plugins/favorites
[2] https://pub.dev/packages/geolocator

Database For storing persistent data, Flutter recommends two solutions: a library offering object relational mapping (ORM) on top of SQLite called *Drift*[3] and a NoSQL database called *Hive*. Since the data model of the HS application fits the relational model nicely and because SQLite is a well known and very high quality embedded database, the first option, *Drift*, was explored initially. On one hand, as an ORM, *Drift* offers a good integration of the SQLite database in a Dart application. For this it uses code generation based on the database models specified directly with Dart code. Thus, it allows for type safe interaction with the SQLite database. On the other hand, it comes with all the usual complexity of an ORM and requires a non-negligible development effort to figure out how to query the models through the ORM APIs. It does however allow one to specify certain queries directly in SQL, for which it then creates appropriate type-safe glue code. *Drift* is also a package designated as a *Flutter Favorite*.

Bluetooth For the functionality of interacting with the Bluetooth system, there was unfortunately no package that met the quality standards of the *Flutter Favorite Program*. In the package repository[4] for Dart and Flutter one can find more than twenty packages for interacting with Bluetooth in various states of abandonment. Since most of these packages have not been updated in quite some time, they are unusable in a new Flutter codebase. This is mostly due the rapid changes in Dart and Flutter which renders older software incompatible quickly. One relatively newer concept in Dart is *Null-safety* [62]. Variables designated as null safe can never be assigned a *null* value. Thus, certain runtime errors can be avoided since a static analyzer prevents the mixing of nullable and non-nullable types in newer Dart versions. However, this makes it difficult to interact with older Dart code which were written with nullable types in their APIs. Most of the Bluetooth packages were therefore ignored since building upon them would have either required migrating the package to null safety or disabling the null safety features for the whole application.

The most popular Bluetooth package for Flutter is called *flutter_blue*. However, it has been abandoned by the original developer about two years ago and therefore has also not received any updates and compatibility fixes for newer Flutter versions. Unfortunately, the original developer did not hand over the package to any new maintainer, with the result that new issues and patches are filed on a regular basis, almost weekly, to report new bugs and to fix older bugs. But, since nobody can update the Github project, it appears that most of the bug reporters or bug-fix contributors are unaware that the package is no longer maintained. A number of forks have emerged, where each new maintainer declared to remedy the situation, integrate all new patches and maintain the package moving forwards. The most healthy of these forks appeared to be the *flutter_blue_plus* package [5].

[3] https://pub.dev/packages/drift
[4] https://pub.dev/
[5] https://pub.dev/packages/flutter_blue_plus

5.2 Development workflow

The source code was hosted in a GitHub repository[6]. A GitHub action was used for continuous integration. On a push or pull request to either the master or develop branch, a static analysis of the source code was performed (also known as linting) using the builtin tool `flutter analyze`. This guaranteed that the shared source code was always in a state that would at least compile. After the linting a small set of unit tests was run, which verified that the parsing of device data from the BLE system and certain interactions with the database were always working as intended. The UI was not tested using unit tests. A Kanban board hosted on Trello was used to keep track of tasks.

5.3 Android

The primary development target for HS is the Android platform (see subsection 4.1.1). During development and testing of the scanner component numerous issues specific to Android showed up.

5.3.1 Background Location Acquisition

The scanner component first tries to acquire the physical location of the device to register a *waypoint* and then scans for BLE devices nearby. As specified in the requirements (subsection 4.1.1) this should be possible to perform in the background. Initially the *workmanager* plugin[7] was used to run the scanner component independently from the UI. This plugin works by using the *WorkManager* functionality on the Android platform[8]. The acquisition of the GPS location worked at first in the background after obtaining all required Android permissions for the application. However after some days of testing and usage in the background, the execution of the scanner component suddenly stopped without any exception thrown at the Dart/Flutter level. This posed some difficulties for two reasons, first code executed with the *workmanager* is unreliable to debug in *AndroidStudio*, because execution seemed to no stop reliably at breakpoints for unknown reasons. It is not clear whether this is a fundamental limitation, a bug in the used Flutter/Android/AndroidStudio combination or whether certain additional Android/Flutter API calls are needed to facilitate debugging in this case. These complications might be related to the fact that code executed within *workmanager* runs in a separate *Dart engine* with no direct way to communicate with the main application code which runs in the Flutter UI engine. A second issue is that Android limits the execution in the background using *workmanager* to at most once every 15 minutes, but the timing is not guaranteed, thus the code might be necessarily executed exactly every 15 minutes. Thus, investigating the issue of no longer working GPS location acquisition was very tedious.

[6] https://github.com/home-scout-project/app
[7] https://pub.dev/packages/workmanager
[8] https://developer.android.com/topic/libraries/architecture/workmanager

It turned out that there are multiple power-saving modes on Android some from stock Android and others specific to the respective smartphone manufacturer. Depending on the particular smartphone these modes are called "Optimize battery usage", "Auto optimize daily" or "Adaptive battery". When one of these modes was activated for the HomeScout application, the background execution was suddenly limited or completely halted. These power-saving modes would simply block the application during acquisition of the GPS location until the application returned to the foreground again. The issue with these modes is that they can kick in without any notification to the user, based on some heuristic which is dependent on the phone manufacturer. It is apparently not possible to ask for exceptions with code similar to Android's permission system. Whether the occurrence of this can be detected by the application itself was not investigated. To resolve the issue, the HomeScout application had to be manually exempted from these power-saving modes in the Android system settings. This appears to be a common issue for background execution in Android phones from different manufacturers[9].

As an additional resolution for difficult to diagnose issues in the scanner component, the background execution was switched from using the *workmanager* to using *flutter_foreground_task*[10]. On Android, compared to *workmanager*, where a specific function which is scheduled to run in the background, is executed in a different process, *flutter_foreground_task* allows to run specific Dart functions within the same *Dart engine*, but in a different *Dart isolate*. A *Dart isolate* allows the single threaded execution of a piece of Dart code which shares no mutable state with other isolates. Message passing allows communication between different *Dart isolates* [63]. The advantage of using *flutter_foreground_task* is that it allows to schedule recurring tasks at any interval, compared to the 15 minutes limit of *workmanager*. The disadvantage is that the recurring task is essentially running in the HomeScout application process, thus if the application is closed then the scanning stops as well. Initially this type of background execution was planned only for the use of the *Heightened Awareness Mode* which tries to scan more aggressively than possible with the *workmanager*, but after discovering the issues mentioned above, the *Normal Mode* also uses the *flutter_foreground_task*.

A hybrid of both execution modes was considered and the implementation of HomeScout allows running the scanner with both modes by setting a specific flag in the code. However this combination has additional drawbacks because it hits another set of limitations from Android. Starting with version 8 (API level 26) the acquisition of a GPS location is limited to "[...] only a few times each hour", the precise numerical limits are not public [64]. Thus when scanning at a certain frequency the request for the GPS locations are occasionally paused.

[9] https://issuetracker.google.com/issues/122098785
[10] https://pub.dev/packages/flutter_foreground_task

5.3.2 Background Bluetooth Scanning

Another major issue on Android was the scanning for Bluetooth devices in the background with either of the two methods as described in the previous section. After the scanning component was implemented in the UI *Dart isolate* and working at the press of a button, the calls to this component were moved into the background. However this did not simply work out of the box. This appeared to be a frequently raised, but still unsolved issue in both the original *flutter_blue*[11] as well as the newer fork *flutter_blue_plus*[12]. Upon closer inspection it was discovered that there is another limitation from the Android power-management system that results in the suspension of execution when scanning for Bluetooth devices in the background and when the device screen is off, which resumes after the application returns into the foreground. However, at that point, the Bluetooth subsystem returns data to the Dart plugin after it has already closed certain resource handles and the scan results do not necessarily reach the Flutter application anymore. Similar to the issue with the GPS location request in the background, the Bluethooth scan with screen off does not lead to an exception or similar notice, instead the whole Flutter engine appears to be just paused. Here, the limitation is that recent Android versions disallow an unfiltered Bluetooth scan in the background, presumably to save power. To receive Bluetooth scan results in the background, a *ScanFilter* has to be used from Android's BLE API [65]. With a *ScanFilter* the BLE subsystem only returns those scan results to the Flutter application, that match the filter. Results can be filtered for names, Service UUIDs, MAC addresses or some other fields.

Although the flutter plugin used for the Bluetooth functionality (*flutter_blue_plus*) did provide support to filter for known MAC addresses and for *Service UUIDs*, it allows only one type of filter at the same time. For the use case of HomeScout, multiple types of *ScanFilters* are essential, using a MAC address it would be possible to look for previously seen devices however this way new devices could only be detected when the screen is on. Alternatively with a *ScanFilter* on *Service UUIDs* the *Tile* (subsection 3.2.2) and *Samsung SmartTag* (subsection 3.2.4) trackers could be detected. To allow detection of these two trackers in the background, the source code of the *flutter_blue_plus* plugin had to be patched. The changes that fixes this bug and allows for setting multiple *ScanFilter* types at the same time was contributed back to the open source community with a pull request[13]. However, at the time of writing the maintainer of the plugin has not yet resumed activity and thus neither integrated nor interacted in any way with the pull request. Thus the HomeScout application uses the forked version of the *flutter_blue_plus* plugin.

Unlike the *Tile* or *Samsung SmartTag* devices, which publish information about their kind through *Service UUIDs*, an *AirTag* can only be identified through the *Manufacturer Data* field (see subsection 3.2.3). Unfortunately, *flutter_blue_plus* did not expose any API to enable a *ScanFilter* on the *Manufacturer Data*. Therefore another patch for the *flutter_blue_plus* plugin was developed. Unlike the first patch, which only dealt with a bug in the *Dart* code, the changes for the *Manufacturer Data* required modifications in the *Dart* interface, the Java implementation and the serialization between the two languages

[11] see issues: #113, #140, #663, #900, #1070 and #1112 in *flutter_blue*
[12] see issues: #26, #74 and #167 in *flutter_blue_plus*
[13] https://github.com/boskokg/flutter_blue_plus/pull/173

using *Protocol Buffers*[14]. These changes were again contributed back to the open source plugin with a pull request[15]. With this patch it is possible to enable a *ScanFilter* for devices manufactured by Apple which are part of the *Find My* network, by setting a byte mask to the appropriate values on the *Manufacturer Data* field. As with the first patch, these changes have not been reviewed nor integrated by the maintainer of the plugin.

5.4 iOS

Although iOS was not the primary target as specified in the requirements (subsection 4.1.1), the HomeScout application was also developed to work for iOS. For the most part, only smaller fixes were needed to get certain things working for iOS. A handful of errors only manifested themselves in release builds, but not in debug builds. As such, they were particularly hard to debug. Most of these errors originated from issues with isolates.

Due to limitations imposed by iOS, *flutter_foreground_task* was not in practice able to reliably perform background tasks. While it was more robust using the *workmanager* plugin, as the overall experience with *flutter_foreground_task* was preferable, as well as because of the benefits of using a unified system across systems, it was decided to use *flutter_foreground_task* on iOS as well. However, as previously mentioned, *workmaneger* still remains included in the application, behind a flag.

5.5 Application Scenario Parameters

5.5.1 Location Designation

As described in section 4.2.3, we introduce the concept of saving certain locations as either safe or dangerous. This data is then taken into account during threat detection, i.e. being in a safe location generally reduces the threat likelihood, whereas being in an dangerous one increases it. The exact parameters are as follows: Safe locations are used in the threat assessment to immediately classify a tracker as a non-threat, if a tracker is detected while a user is in a safe location and in normal mode. If a user is in a dangerous area, the threshold values needed to classify a device as a threat are lowered, i.e. a device is more quickly classified as a threat. In heightened awareness mode, if a user is in a dangerous location, it too lowers the threshold values for a device to be considered a threat.

[14] https://developers.google.com/protocol-buffers/
[15] https://github.com/boskokg/flutter_blue_plus/pull/195

5.5.2 Device Block- and Allowlisting

Section 4.2.4 described the ability to mark devices as being familiar or unknown, or in more technical terms to allowlist or blocklist a device. This information is then taken into account by the threat classification algorithm. In the normal running mode, if an allowlisted tracker is detected it is immediately classified as a non-threat. In heightened awareness mode however, even allowlisted devices are not immediately discounted from being a threat, but must go through the rest of the checks as well.

A blocklisted device in heightened awareness mode will lead to a device immediately being classified as a threat on a subsequent detection. In normal running mode it will not do so immediately, but it does increase the likelihood of it being classified as a threat.

Chapter 6

Evaluation

To perform the experiments, a special experiment mode was added to the HomeScout application. In this mode the threat analysis component only considers data gathered within the experiment's lifetime. This allows for efficient data collection under various conditions. As described in detail in section 4.6, for each experiment run, the experiment mode was enabled prior to enabling the scanner. The first scan performed is therefore performed at t_0. The experiments were conducted with multiple smartphones and different numbers and types of trackers. Prior to the analysis, the data from each smartphone was pooled into one dataset. The data was filtered for experiments which contained at least two scans, which removes spurious data where the experiment toggle button was activated only briefly without the intention to actually perform an experiment.

6.1 Experiment 1: Tracker Detection

For the first experiment, we collected 21 data traces in the *Heightened Awareness Mode* and 7 in the *Normal Mode*. Descriptive statistics about the data collected in each mode is shown in Table 6.1 and Table 6.2. On average, a potentially malicious BLE tracker was detected after 4.75 minutes in the *Heightened Awareness Mode* compared to 60.5 minutes in the *Normal Mode*.

Table 6.1: Descriptive statistics for runs in Heightened Awareness Mode ($N = 21$)

Statistic	Mean	St. Dev.	Min	Max
Duration [min]	12.4	9.9	3.5	36.6
Number of BLE scans performed	10.8	10.0	3	35
Time to first tracker detection [s]	285.2	211.8	13	737
Distance to first tracker detection [m]	229.0	160.7	110	658
Average Nr. of Devices observed per scan	16.0	24.2	0.8	86.4
Average Nr. Trackers observed per scan	2.3	1.2	0.8	5.8
Average Nr. Threats observed per scan	3.4	6.5	0.0	29.6

Table 6.2: Descriptive statistics for runs in Normal Mode ($N = 7$)

Statistic	Mean	St. Dev.	Min	Max
Duration [min]	113.3	59.9	19.1	191.2
Number of BLE scans performed	8.4	2.4	6	13
Time to first tracker detection [s]	3,631.6	2,970.6	1,015	9,918
Distance to first tracker detection [m]	1,537.6	1,171.4	138	3,472
Average Nr. of Devices observed per scan	13.2	9.0	6.3	31.4
Average Nr. Trackers observed per scan	2.0	1.1	0.9	3.8
Average Nr. Threats observed per scan	1.4	1.0	0.2	3.2

6.1.1 Detection Rate

Several measures have been computed for the estimation of the detection rate, which are shown in Table 6.3. The first statistic is (`detection_rate_experiment`), which measures the ratio of experiments that ended up in an alert for a tracker versus those that did not. It is very high for both modes, but this result is heavily dependent on the duration of each experiment. The longer an experiment took, the higher the probability that a tracker can be detected over the whole length of the experiment.

Table 6.3: Detection rate estimates for both scan modes

Statistic	Normal Mode	Heightened Awareness Mode
`detection_rate_experiment`	0.58	0.80
`detection_rate_fast`	0.34	0.5
`detection_rate_t15m` [s]	-	0.76

The second statistic (`detection_rate_fast`), measures the ratio of experiments that detected a tracker within the shortest possible time, given the specific detection parameters used in the tracker detection algorithm. Only a small fraction of the experiment runs in the *Heightened Awareness Mode* resulted in the fastest possible detection at the third scan. Most of the detections occurred on the fourth scan as can be seen in Figure 6.1, which shows a histogram of the number of scans performed before the tracker was detected. In the *Normal Mode* no tracker was detected at this threshold, which is expected since as seen with the *Heightened Awareness Mode*, not every scan discovers all carried devices and usually at least four scans are required. Furthermore the sample size in the *Normal Mode* is lower, thus if more data would be collected, this detection rate would presumably be non-zero.

The third statistic (`detection_rate_t15m`), is closely related to the second one, but with the unit of time instead of number of scans. Also, the threshold is more relaxed compared to `detection_rate_fast`. Since the *Normal Mode* has a scan interval of 15 minutes, this statistics cannot be computed for that mode. For the *Heightened Awareness Mode* the results show that a potentially malicious BLE tracker is detected within 15 minutes in 89% of the experiments.

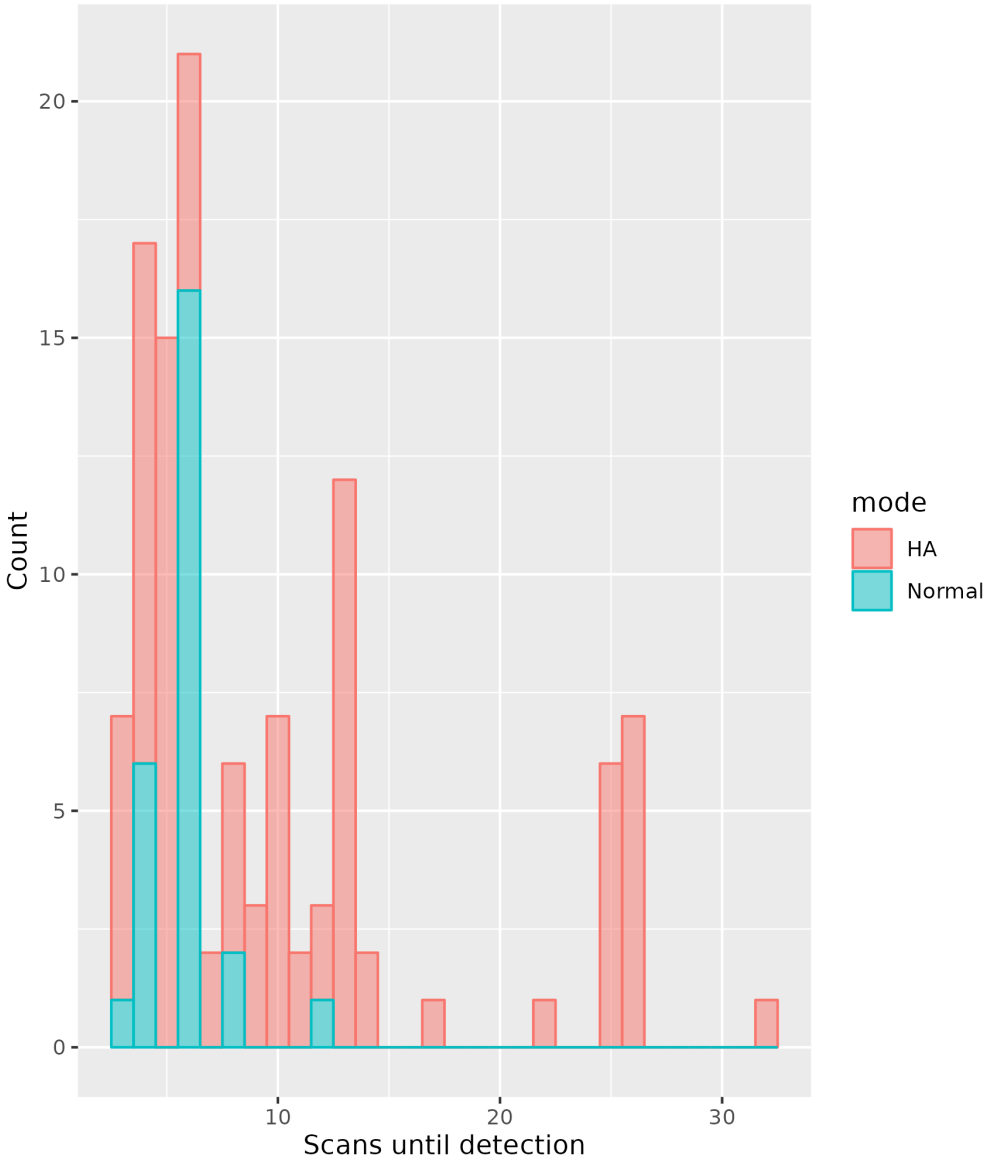


Figure 6.1: Distribution of required scans to detect a tracker

6.1.2 Tracker Differences

Depending on the availability, the experiment runs were carried out with multiple trackers at the same time. This allows to compare the detection rates for different trackers. For the *Heightened Awareness Mode* the number of detected devices for each device type per experiment run are shown in Figure 6.2. Since at most only one device of each device type was carried during the experiment, it can be assumed for some device types, that the additionally detected devices are false positives. Particularly for the *Tile* trackers, this allows for the estimation of false positive rate, since their MAC address does not change, unlike the *AirTags* or *SmartTags*. For these devices additional detections might be a false positive or simply the same tracker detected multiple times, by observing it during the MAC randomization cycle.

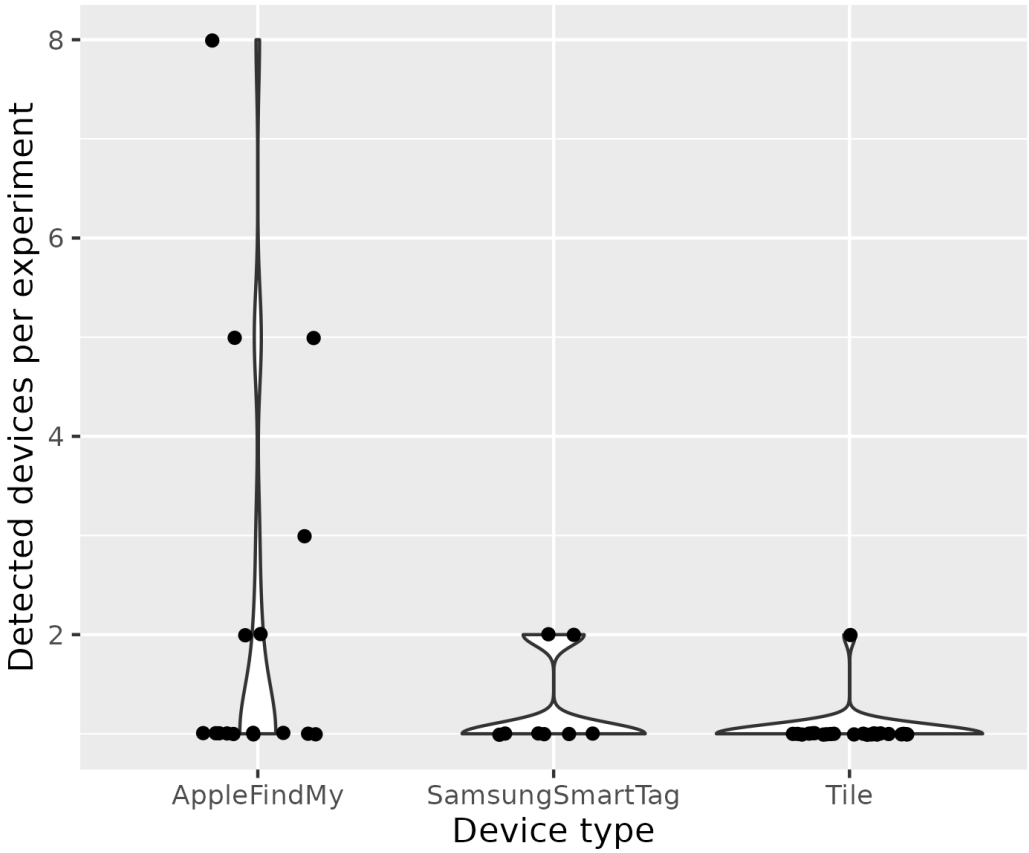


Figure 6.2: Distribution of number of detected devices per device type for all experiments

With the simplifying assumption that all additionally detected unique MAC addresses were not the ones carried during the experiment. An estimation of the false alarm rate can be made. In Table 6.4 a false alarm rate was computed as the rate of experiments with extra devices detected to the number of total experiments for each device type and scan mode.

The false alarm rate is the highest for devices identified as participating in the *Apple Find My* network, presumably *AirTags*, which can also be seen in Figure 6.2. This might be due to the popularity of *AirTags* compared to the other device types, in that there are simply more of the Bluetooth trackers used by the general public.

Table 6.4: Detection rate estimates

Scan Mode	Device Type	Number of experiments	False Alarm ratio
HA	AppleFindMy	10	0.50
HA	SamsungSmartTag	8	0.25
HA	Tile	16	0.00
Normal	AppleFindMy	2	0.00
Normal	Tile	4	0.25

Another interesting insight is that the *Samsung SmartTag* was not detected in the *Normal Mode*. It was hypothesized that this might be due to MAC address randomization at a faster interval than the scan interval of the *Normal Mode*. This behaviour has previously been reported with an interval of approximately 15 minutes for certain BLE advertisement data such as the Samsung specific *Agin Counter, Privacy ID* and *Signature* fields [66]. However the exact MAC address rotation for *Samsung SmartTags* depends on the separation state and separation duration, allegedly it is however similar to the behaviour of *AirTags* [66].

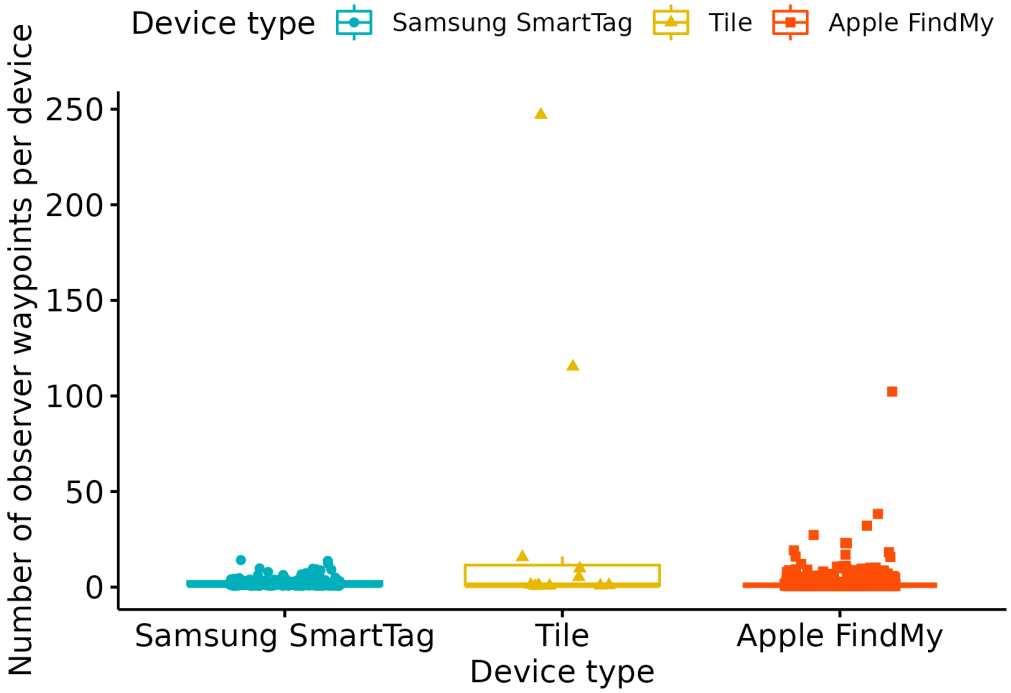
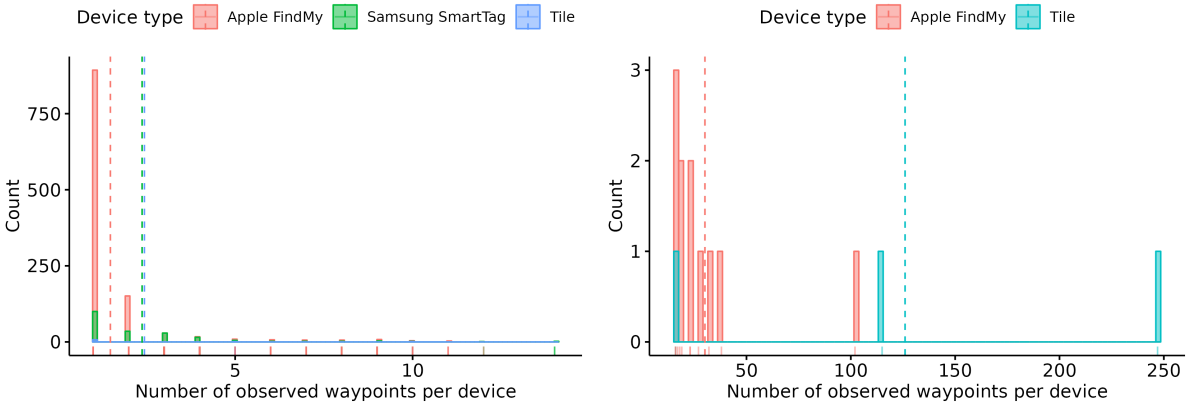


Figure 6.3: Distribution of number of observations per device address

From collected data of the experiments described in this report, the behaviour of the *Samsung SmartTags* can be further investigated. In Figure 6.3, the distribution of the number of waypoints at which a particular device was observed, identified by its MAC address, across multiple experiments was plotted. For the *Tile* trackers, the two outlier data points represent the two *Tile* trackers used in this work. Because they do not change their MAC address, their identification is trivial.

If *Samsung SmartTags* change their MAC address at a frequency of 15 minutes, then no device should be detected for a longer period. This can be observed in Figure 6.4, which shows the distribution of the number of observations at a distinct waypoint for each observed unique device. For better visibility the histogram was split into two figures, where the first one (Figure 6.4a) shows only devices which have been observed for at most 15 waypoints, the second (Figure 6.4b) shows devices which were observed for more waypoints. In the *Heightened Awareness Mode* a waypoint occurs at approximately every minute, thus the horizontal axis can be interpreted as the duration in minutes over which a particular device has been observed. From Figure 6.4a it is apparent, that no *Samsung SmartTag* was observed for more than 15 minutes.



(a) Number of observed waypoints ≤ 15 (b) Number of observed waypoints > 15

Figure 6.4: Distribution of number of observations per device address

The rotation of the MAC address is supposedly recommended in the Bluetooth Core Specification [28] to prevent tracking of a BLE device [67]. Conversely, Samsung's compliance with this recommendation for their *SmartTag* products leads to increased risks from malicious usage of these trackers. As discussed in subsection 5.3.1, this is relevant especially for Android smartphones, where the usual background scanning is limited to 15 minutes [64]. Other anti-tracking solutions such as *AirGuard*, which use the *WorkManager* from Android, are also limited to the 15 minuute scan interval [12].

6.2 Experiment 2: Allowlisting

The second experiment provides data on the performance of the allowlist of HomeScout application. Allow list is a dynamic library that can add the specific device manually by the user to avoid ominous messages. The experiment was run three times with several trackers to validate the functionality of the allowlist. In all three runs, the user found no alerts as the trackers were no longer detected, once the trackers were added to the allowlist. This indicates that the allowlist was effectively ignored trackers that had already been added. After being once again removed from the allowlist, one tracker continued to not show alerts.In two out of three experiment runs, the user found alerts for the device that have been deleted successfully, however, further analysis is needed to determine the cause of the false negative.

Table 6.5: Experiment 2: Allowlisting

Duration(s)	Distance(m)	Number of trackers added to the allowlist	User finds no alerts for allowlisted device (Y/N)	User finds alerts for device deleted from the allow list (Y/N)
1320	1846	6	Y	Y
732	1040	4	Y	N
505	717	3	Y	Y

6.3 Experiment 3: Blocklisting

The third experiment is similar to the previous one, which also consists of three trials performed with multiple trackers. The block list is also a dynamic library but it enables users to add devices that expecting having more alerts. The first and the second column represents the time and the distance taken for each trial, and the third column records the number of trackers added to the block list with an average of 4 trackers added per experiment approximately. And the last column presents the outcome of whether the user received a quick notification upon detecting previously blocklisted devices. Across all trackers and trials, trackers were more quickly classified as a threat after being added to the blocklist.

Table 6.6: Experiment 3: Blocklisting

Duration(s)	Distance(m)	Number of trackers added to the block list	User finds quick alerts of previously blocklisted device having been detected(Y/N)
747	1060	5	Y
672	940	3	Y
702	985	4	Y

6.4 Experiment 4: False alarms

This experiment studied the false alarm rate of the HomeScout application when there is no known trackers are in proximity. The quantitative data was collected, involved in three trials, and the result of each trial contains three variables: duration, distance, the number of BLE scans performed in total and the number of false alarms received. Across all trials, users get alerts without taking any trackers themselves. It is worth raising concerns that false alarms can have crucial an impact on the reliability of the tracking device as it may lead to unnecessary interruption for users which will be considered annoying or disruptive when using the application.

Table 6.7: Experiment 4: False alarms

Duration(s)	Distance(m)	Number of BLE scans performed	Number of false alarms received
347	485	4	1
586	832	10	2
457	650	8	2

6.5 Experiment 5: Location Designation

The last experiment aims to access the efficacy of the geofencing component in tracking devices and to examine the users'ability to designate specific locations as either safe or unsafe depending on the high or low risk of being tracked. The experiment consists of two rounds for each trial, with data collected in terms of location label, time to tracker detected and the number of the trackers detected in the first and second round.

In the first trial, the time taken to detect the same amount of trackers in *Unsafe* location were faster than the location was designated as *Unsafe*. The second and the third trial were showing the opposite, which is the time taken to detect the same number of trackers in *Safe* location is longer than before the location was designated as *Safe*. Therefore, the result indicates that the users are able to designate certain locations as high or low risk of being tracked.

Table 6.8: Experiment 5: Location designation

Location Label(Safe or Unsafe)	Time to trackers detected during experiment round 1(s)	Number of first tracker detection during experiment round 1	Time to trackers detected during experiment round 2(s)	Number of trackers detected during experiment round 2
Unsafe	720	2	580	2
Safe	467	1	747	0
Safe	641	1	966	1

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this paper it was examined, how BLE devices can be used beyond their originally designed purpose of tracking lost items. Previous work demonstrates, how smart trackers can be utilized as crowd-sensing devices and the limitations of products utilizing different platforms have also been discussed, which leads to raising awareness of such possibilities to design an extensible modular application that operates across multiple platforms.

In consequence, this paper proceeded to introduce HomeScout, a cross-platform, smartphone-based BLE counter-surveillance solution. The purpose of HS is to provide users with a reliable and effective way of detecting potential malicious tracking from nearby Bluetooth trackers. To achieve this goal, the tracker detection and classification algorithm was improved and implemented into HS. The algorithm was based on four use-case scenarios that provide users with a better understanding of interacting with the system and raising concerns about potential threats. Throughout the design and development phase, the limitations of a smartphone application were considered and tested in a series of experiments. To measure the performance outcome of the algorithm, a validation dataset was collected and analyzed for later evaluation. The results of this evaluation analysis showcased that HS is considered to be effective in detecting potential threats, as it provides users with actionable alerts whether the tracking device is assumed as *dangerous* or *non-dangerous*.

The HS application improves upon existing alternative solution in multiple ways. First, it considers four tracking scenarios, with modifiable thresholds to adapt to each of them differently, whereas other anti-tracking solutions provide only a „one size fits all“ threshold. Second, it offers the technical framework to perform scans at any time interval, by using all available APIs offered by the Flutter platform, whereas other solutions are limited to at most 15 minutes intervals. Third, it provides a special experiment mode, which allows easy collection of real-world data in the field, next to providing protection from trackers. And fourth, it fixes and extends the capabilities of the Flutter BLE driver plugin, allowing scanning in the background as well as more efficient scanning overall, by filtering for potentially malicious devices at the BLE driver level directly, without having to process all available devices in user space.

7.2 Future Work

The current implementation of HS serves as a proof-of-concept, drawing on the research outcomes of related papers. It solves the purpose of performing quick detections across different platforms through the use of an advanced device detection and classification algorithm. However, it should be noted that the current development is not completed and does not include all the features that a such a system could possess.

In the future, the design of HS could aim to enhance its functionality and provide a more comprehensive experience for users. For instance, the integration of a map to show tracker detections, marked locations and more. Future iterations of the work could prioritize the incorporation of increased customization options to better meet individual needs. Additionally, an effort could be made to improve compatibility with a wider range of devices and platforms. In particular, the limitations imposed by tracker manufacturers such as MAC randomization, would be of interest. While HS did a first step in designing, refining and testing certain parameters and algorithmic flows, this could be further refined through large-scale tests and feedback from users. Lastly, additional use-case scenarios could be conceptualized and implemented, to further cover the situations in which users might need an application such as HomeScout.

Two specific avenues for future investigations are some sort of heuristic device matching, which tries to match different observed MAC addresses to the same physical device, when it undergoes MAC address randomization. In the case of the *Samsung SmartTag* the device identifier as advertised in the service data rotates not at the same time as the MAC address, thus at least two consecutive MAC addresses could be tied to the same device. Then, depending on the occurrences of other devices, certain guesses could be made that two MAC addresses are in fact the same device if they are non-overlapping in time, but observed at the same rate from the moving user of HS.

Another interesting avenue, would be to use the information contained in the RSSI to create a device type specific metric to estimate the actual distance of the device to the smartphone. It might be possible to calibrate such a metric for a range of devices and thus improve the device distance traveled measure.

Bibliography

[1] Asset Infinity, "Reasons why you should choose ble asset tracking over rfid asset tracking," Sep 2022. [Online]. Available: https://www.assetinfinity.com/blog/why-choose-ble-asset-tracking-over-rfid-asset-tracking

[2] "Ble asset tracking - what you need to know - bluetooth ble tags," Jan 2023. [Online]. Available: https://radiantrfid.com/blog/ble-asset-tracking-what-you-need-to-know/

[3] IEEE, *IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture*, 2002.

[4] N. Mitro, M. Krommyda, and A. Amditis, "Smart tags: Iot sensors for monitoring the micro-climate of cultural heritage monuments," *Applied Sciences*, vol. 12, no. 5, 2022. [Online]. Available: https://www.mdpi.com/2076-3417/12/5/2315

[5] R. Mac and K. Hill, "Are apple airtags being used to track people and steal cars?" Dec 2021. [Online]. Available: https://www.nytimes.com/2021/12/30/technology/apple-airtags-tracking-stalking.html

[6] K. Hill and P. T. Heisler, "I used apple airtags, tiles and a gps tracker to watch my husband's every move," Feb 2022. [Online]. Available: https://www.nytimes.com/2022/02/11/technology/airtags-gps-surveillance.html

[7] M. Allison, "Apple's airtags keep being tagged in domestic abuse cases," Apr 2022. [Online]. Available: https://www.digitaltrends.com/mobile/apple-airtag-domestic-abuse-police-reports/

[8] A. Heinrich, M. Stute, T. Kornhuber, and M. Hollick, "Who can find my devices? security and privacy of apple's crowd-sourced bluetooth location tracking system," *Proceedings on Privacy Enhancing Technologies*, vol. 2021, pp. 227–245, 07 2021.

[9] Chipolo, "Chipolo one spot: How it works," 2022. [Online]. Available: https://chipolo.net/en/blogs/chipolo-one-spot-how-it-works

[10] A. Johnson, "The search is over: Smart trackers from apple, samsung, and tile compared," Jul 2021. [Online]. Available: https://www.theverge.com/22570161/apple-airtag-samsung-smarttag-tile-pro-bluetooth-tracker-review-test-comparison

[11] K. Balasaygun, "The biggest risks of using bluetooth trackers like apple airtag, tile," Jan 2023. [Online]. Available: https://www.cnbc.com/2023/01/14/the-biggest-security-pros-and-cons-of-using-bluetooth-gps-trackers.html

[12] A. Heinrich, N. Bittner, and M. Hollick, "Airguard-protecting android users from stalking attacks by apple find my devices," in *Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2022, pp. 26–38.

[13] A. Ozer and E. John, "Improving the accuracy of bluetooth low energy indoor positioning system using kalman filtering," in *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2016, pp. 180–185.

[14] D. Chen, K. G. Shin, Y. Jiang, and K.-H. Kim, "Locating and tracking ble beacons with smartphones," in *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 263–275. [Online]. Available: https://doi.org/10.1145/3143361.3143385

[15] Y.-J. Lin, H.-S. Chen, and M.-J. Su, "A cloud based bluetooth low energy tracking system for dementia patients," in *2015 Eighth International Conference on Mobile Computing and Ubiquitous Networking (ICMU)*, 2015, pp. 88–89.

[16] J. Kolakowski, V. Djaja-Josko, M. Kolakowski, and K. Broczek, "Uwb/ble tracking system for elderly people monitoring," *Sensors*, vol. 20, no. 6, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/6/1574

[17] A. K. Das, P. H. Pathak, C.-N. Chuah, and P. Mohapatra, "Uncovering privacy leakage in ble network traffic of wearable fitness trackers," in *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications*, ser. HotMobile '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 99–104. [Online]. Available: https://doi.org/10.1145/2873587.2873594

[18] H. Givehchian, N. Bhaskar, E. R. Herrera, H. R. L. Soto, C. Dameff, D. Bharadia, and A. Schulman, "Evaluating physical-layer ble location tracking attacks on mobile devices," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 1690–1704.

[19] L. Tonetto, A. Carrara, A. Y. Ding, and J. Ott, "Where is my tag? unveiling alternative uses of the apple findmy service," in *2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2022, pp. 396–405.

[20] A. Heinrich, M. Stute, T. Kornhuber, and M. Hollick, "Who Can Find My Devices? Security and Privacy of Apple's Crowd-Sourced Bluetooth Location Tracking System," Mar. 2021.

[21] A. Heinrich, M. Stute, and M. Hollick, "Openhaystack: A framework for tracking personal bluetooth devices via apple's massive find my network," in *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, ser. WiSec '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 374–376. [Online]. Available: https://doi.org/10.1145/3448300.3468251

[22] J. Martin, D. Alpuche, K. Bodeman, L. Brown, E. Fenske, L. Foppe, T. Mayberry, E. Rye, B. Sipes, and S. Teplov, "Handoff all your privacy – a review of apple's bluetooth low energy continuity protocol," *Proceedings on Privacy Enhancing Technologies*, vol. 2019, pp. 34–53, 10 2019.

[23] T. Mayberry, E. Fenske, D. Brown, J. Martin, C. Fossaceca, E. C. Rye, S. Teplov, and L. Foppe, "Who tracks the trackers? circumventing apple's anti-tracking alerts in the find my network," in *Proceedings of the 20th Workshop on Workshop on Privacy in the Electronic Society*, ser. WPES '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 181–186. [Online]. Available: https://doi.org/10.1145/3463676.3485616

[24] T. Mayberry, E.-O. Blass, and E. Fenske, "Blind my - an improved cryptographic protocol to prevent stalking in apple's find my network," *Proceedings on Privacy Enhancing Technologies*, vol. 2023, no. 1, pp. 85–97, 2023.

[25] M. Weller, J. Classen, F. Ullrich, D. Waßmann, and E. Tews, "Lost and found: Stopping bluetooth finders from leaking private information," in *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. Linz Austria: ACM, Jul. 2020, pp. 184–194.

[26] C. Garg, A. Machiry, A. Continella, C. Kruegel, and G. Vigna, "Toward a secure crowdsourced location tracking system," in *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, ser. WiSec '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 311–322. [Online]. Available: https://doi.org/10.1145/3448300.3467821

[27] J. Briggs and C. Geeng, "Ble-doubt: Smartphone-based detection of malicious bluetooth trackers," in *2022 IEEE Security and Privacy Workshops (SPW)*, 2022, pp. 208–214.

[28] *Bluetooth Core Specification*, Bluetooth SIG, 2019, rev. 5.2.

[29] A. Catley, "Apple AirTag Reverse Engineering," https://adamcatley.com/AirTag.html, 2022.

[30] T. Roth, F. Freyer, M. Hollick, and J. Classen, "AirTag of the Clones: Shenanigans with Liberated Item Finders," in *2022 IEEE Security and Privacy Workshops (SPW)*. San Francisco, CA, USA: IEEE, May 2022, pp. 301–311.

[31] M. Woolley, "The Bluetooth® Low Energy Primer," Tech. Rep., 2022.

[32] A. Tanenbaum, *Computer Networks*, 3rd ed. Pearson Prentice Hall, 1996.

[33] F. Forno, G. Malnati, and G. Portelli, "Design and implementation of a Bluetooth ad hoc network for indoor positioning," *IEE Proceedings - Software*, vol. 152, no. 5, p. 223, 2005.

[34] D. Schwarz, M. Schwarz, J. Stückler, and S. Behnke, "Cosero, Find My Keys! Object Localization and Retrieval Using Bluetooth Low Energy Tags," in *RoboCup 2014: Robot World Cup XVIII*, R. A. C. Bianchi, H. L. Akin, S. Ramamoorthy, and K. Sugiura, Eds. Cham: Springer International Publishing, 2015, vol. 8992, pp. 195–206.

[35] N. Lomas, "Tile Wants You To Stop Losing Important Stuff With Its Bluetooth Tags Plus App Combo," 2013. [Online]. Available: https://techcrunch.com/2013/06/20/tile/

[36] Amazon, "Echo, Tile, and Level devices join Amazon Sidewalk," 2021. [Online]. Available: https://www.aboutamazon.com/news/devices/echo-tile-and-level-devices-join-amazon-sidewalk

[37] Apple, "Apple introduces airtag," Apple, 2021. [Online]. Available: https://www.apple.com/newsroom/2021/04/apple-introduces-airtag/

[38] *Assigned Numbers*, Bluetooth SIG, 2023.

[39] *Supplement to the Bluetooth Core Specification*, Bluetooth SIG, 2021, rev. 10.

[40] Samsung, "[Update] Introducing the New Galaxy SmartTag+: The Smart Way to Find Lost Items," 2021. [Online]. Available: https://news.samsung.com/us/introducing-the-new-galaxy-smarttag-plus/

[41] B. K, "Samsung galaxy smarttag+ to use ar to visually locate your missing items," Apr 2021. [Online]. Available: https://www.techradar.com/news/samsung-galaxy-smarttag-to-use-ar-to-visually-locate-your-missing-items

[42] Samsung, "Samsung SmartThings Find Hits New Milestone With 200 Million Nodes Helping Find Lost Devices," 2022. [Online]. Available: https://news.samsung.com/us/smartthings-find-milestone-nodes-devices/

[43] D. Thomas, "How to enable offline finding on your galaxy so you can locate your phone in airplane mode," Dec 2020. [Online]. Available: https://android.gadgethacks.com/how-to/enable-offline-finding-your-galaxy-so-you-can-locate-your-phone-airplane-mode-0384159/

[44] H. Jin, "Who tracks who? an surveillance capitalist examination of commercial bluetooth tracking networks," *arXiv preprint arXiv:2211.11070*, 2022.

[45] A. Greenberg, "The clever cryptography behind apple's 'find my' feature," Jun 2019. [Online]. Available: https://www.wired.com/story/apple-find-my-cryptography-bluetooth/

[46] Chipolo, "What's the difference between chipolo one and chipolo one spot?" 2022. [Online]. Available: https://chipolo.net/en/blogs/whats-the-difference-between-chipolo-one-and-chipolo-one-spot

[47] Apple, "Apple's find my network now offers new third-party finding experiences," Nov 2022. [Online]. Available: https://www.apple.com/newsroom/2021/04/apples-find-my-network-now-offers-new-third-party-finding-experiences/

[48] L. Xing, X. Bai, T. Li, X. Wang, K. Chen, X. Liao, S.-M. Hu, and X. Han, "Cracking app isolation on apple: Unauthorized cross-app resource access on mac os x and ios," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 31–43. [Online]. Available: https://doi.org/10.1145/2810103.2813609

[49] Apple, "Use the find my app to locate a missing device or item," Sep 2022. [Online]. Available: https://support.apple.com/en-az/HT210515

[50] ——, "Icloud - find my," 2022. [Online]. Available: https://www.apple.com/icloud/find-my/

[51] J. Briggs and C. Geeng, "Ble-doubt: Smartphone-based detection of malicious bluetooth trackers," in *2022 IEEE Security and Privacy Workshops (SPW)*, 2022, pp. 208–214.

[52] A. Newsroom, "An update on airtag and unwanted tracking," Feb 2022. [Online]. Available: https://www.apple.com/newsroom/2022/02/an-update-on-airtag-and-unwanted-tracking/

[53] A. F. Cahn, "Apple's airtags are a gift to stalkers," May 2021. [Online]. Available: https://www.wired.com/story/opinion-apples-air-tags-are-a-gift-to-stalkers/

[54] K. O. E. Müller, B. Rodrigues, and B. Stiller, "Master project (map) for dylan puser, remo tobias hertig, and zhishan yan," Tech. Rep., 2022.

[55] ISO, *ISO/IEC/IEEE International Standard - Systems and software engineering – Life cycle processes –Requirements engineering*, 2011.

[56] F. Van Diggelen and P. Enge, "The world's first gps mooc and worldwide laboratory using smartphones," in *Proceedings of the 28th international technical meeting of the satellite division of the institute of navigation (ION GNSS+ 2015)*, 2015, pp. 361–369.

[57] B. Eissfeller, A. Teuber, and P. Zucker, "Indoor-gps: Ist der satellitenempfang in gebäuden möglich?" *ZfV-Zeitschrift für Geodäsie, Geoinformation und Landmanagement*, no. zfv 4/2005, 2005.

[58] M. B. Kjærgaard, H. Blunck, T. Godsk, T. Toftkjær, D. L. Christensen, and K. Grønbæk, "Indoor positioning using gps revisited," in *Pervasive Computing: 8th International Conference, Pervasive 2010, Helsinki, Finland, May 17-20, 2010. Proceedings 8*. Springer, 2010, pp. 38–56.

[59] Google Developers, "Guide to app architecture," Android, 2022. [Online]. Available: https://developer.android.com/topic/architecture

[60] E. Gamma, *Design patterns: elements of reusable object-oriented software*. Addison-Wesley, 1995.

[61] Flutter, "Hotfixes to the stable channel," https://github.com/flutter/flutter/wiki/Hotfixes-to-the-Stable-Channel, 2023.

[62] Dart, "Sound null safety," 2022. [Online]. Available: https://dart.dev/null-safety

[63] ——, "Concurrency in dart," 2023. [Online]. Available: https://dart.dev/guides/language/concurrency

[64] Google Developers, "Background location limits," Android, 2022. [Online]. Available: https://developer.android.com/about/versions/oreo/background-location-limits

[65] ——, "Scanfilter," Android, 2022. [Online]. Available: https://developer.android.com/reference/android/bluetooth/le/ScanFilter

[66] T. Yu, J. Henderson, A. Tiu, and T. Haines, "Privacy Analysis of Samsung's Crowd-Sourced Bluetooth Location Tracking System," Oct. 2022.

[67] G. Celosia and M. Cunche, "Saving private addresses: An analysis of privacy issues in the bluetooth-low-energy advertising mechanism," in *Proceedings of the 16th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*. Houston Texas USA: ACM, Nov. 2019, pp. 444–453.

Abbreviations

API Application Programming Interface. 41–45, 57

BLE Bluetooth Low Energy. i, iii, 1–7, 10, 12, 13, 15–17, 20, 21, 23, 24, 38, 43, 45, 49, 50, 53, 54, 57, 67

CRC Cyclic redundancy check. 7

GAP Generic Access Profile. 5, 6

GPS Global Positioning System. 3, 8, 10, 17, 18, 20, 41, 43–45

HS HomeScout. i, iii, 2, 15, 17–19, 21, 25, 33, 34, 36–39, 41–46, 49, 55–58

IoT Internet of Things. 1–3, 5

IPS indoor positioning system. 3

ISA Item Safety Alert. 12

MAC Media Access Control. 1, 11, 45, 51–54, 58

NFC Near-field communication. 10

OF Offline Finding. 11

ORM object relational mapping. 42

PDU Protocol Data Unit. 7, 10

RSSI received signal strength indicator. 3, 8, 13, 58

SoC System-on-Chip. 5, 10

UI user interface. 41, 43, 45

UID Unique Identifier. 1

UWB Ultra-Wideband. 3, 10

Glossary

Allowlist A list of BLE devices which are owned by the user or devices known to be not used for malicious tracking purposes.

Blocklist A list of BLE devices which are deemed untrustworthy or potentially malicious.

Bluetooth tracker A small device containing a battery and a BLE chip, which broadcasts signals according to some offline finding protocol and thus allows the owner of the tracker to monitor the location of the tracker [25, 30].

List of Figures

List of Tables

Appendix A

Installation Guidelines

A.1 Development instructions

1. Install Flutter using the following guide:
   https://docs.flutter.dev/get-started/install

2. Clone the HomeScout source code:
   git clone https://github.com/home-scout-project/app.git

3. Install all dependencies: flutter pub get

4. Run the app on a device: flutter run

A.2 Experiment instructions

Preparation

1. Arm available BLE trackers: put them into *Lost* state

2. Enter *Debug view* (button on bottom right)

3. Verify that the trackers are sending Bluetooth signals by clicking the *Perform device scan* button. The List below should populate with the device details if it's either an AirTag, SmartTag or Tile.

4. Go back to the Homepage

Experiment round

1. Start an Experiment by toggling the *Run Experiment* slider
2. Enable the scanner in the desired mode
3. Perform the experiment: travel around
4. Disable the experiment

Data extraction

1. Connect the phone to a computer with AndroidStudio
2. Open the *Device File Explorer* pane (on bottom right side)
3. Open the HomeScout data folder:
   e.g */data/data/ch.uzh.csg.home_scout/app_flutter*
4. Download the *homescout.sqlite* file (right click: Save as)

Appendix B

Contents of the Zip file

FinalReport.pdf this report

FinalReport folder with the sources for this report

data folder with the raw collected data

analysis folder with scripts used to analyse the data

HomeScout folder with source code of the app

flutter_blue_plus folder with patched version for the dependencies

presentation folder with the slides of the presentations