University of Zurich[UZH]

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

Bluetooth Low Energy Device Classifier

Jie Liao
Zurich, Switzerland
Student ID: 20-740-551

Supervisor: Katharina Müller
Date of Submission: August 8, 2023

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

ifi

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

Zusammenfassung

0.1 Einleitung

Im Jahr 2011 läutete die Einführung der Bluetooth Low Energy (BLE) Technologie eine signifikante Neuerung in der drahtlosen Kommunikation ein. BLE, primär für Energieeffizienz und kurze Distanzkommunikation entwickelt, ebnete den Weg für das Internet der Dinge (IoT). Dies förderte die Entstehung von ortsbasierten Trackern wie Apples AirTag. Obwohl sie nützlich sind, haben sie Sicherheitsbedenken hervorgerufen. Ein Hauptanliegen ist das Risiko, dass Dritte Bewegungen unbemerkt verfolgen könnten. Trotz vorhandener Sicherheitsprotokolle bleiben Fragen bezüglich ihrer Effektivität. Daher ist es essenziell, BLE-Geräte präziser zu identifizieren und zu kategorisieren, um die Sicherheit der Benutzer zu erhöhen.

0.2 Ziele

Das Hauptziel dieser Arbeit ist es, die wachsenden Sicherheits- und Datenschutzbedenken im Zusammenhang mit Trackern und anderen Bluetooth Low Energy (BLE) Geräten zu adressieren. Insbesondere zielt diese Arbeit darauf ab, eine Methode zu entwickeln, die verschiedene BLE-Geräte effektiv klassifizieren kann.

Ein wichtiger Beweggrund für diese Arbeit ist das Ungleichgewicht im Schutz zwischen iOS- und Android-Benutzern. Während iOS-Benutzer von erweiterten Sicherheitsfunktionen profitieren, fehlen Android-Benutzern ähnliche Schutzmassnahmen. Daher konzentriert sich diese Arbeit darauf, eine Anwendung speziell für die Android-Plattform zu erstellen.

Frühere Anwendungen wie HomeScout konzentrierten sich auf die Identifizierung von persönlichen Trackern wie dem AirTag von Apple. Allerdings gibt es bestimmte Einschränkungen in den Klassifikationsfähigkeiten dieser Lösungen. Dieser Forschungszweig zielt darauf ab, diese Grenzen zu überwinden und eine verbesserte Klassifizierungsmethode zu entwickeln, die in die HomeScout-Anwendung integriert wird.

Insgesamt lassen sich die Hauptforschungsziele wie folgt zusammenfassen: Entwicklung einer Methode zur Klassifizierung verschiedener Arten von BLE-Geräten, insbesondere

von Ortungstrackern und anderen IoT-Geräten, sowie Integration des entwickelten Klassifikationsalgorithmus in HomeScout, um dessen Fähigkeit zur Geräteidentifikation über die bloße Erkennung von Trackern hinaus zu erweitern.

0.3 Resultate

Die Einführung von Bluetooth Low Energy (BLE) im Jahr 2011 leitete einen bedeutenden Wendepunkt in der drahtlosen Kommunikation ein und ebnete den Weg für das Internet der Dinge (IoT) sowie den Aufstieg von standortbasierten Trackern. Trotz der Bequemlichkeit, die Geräte wie Apples AirTag bieten, bestehen Sicherheitsrisiken, insbesondere das Risiko, dass böswillige Akteure Personen ohne ihr Wissen verfolgen. Diese Arbeit zielt darauf ab, die Sicherheitsbedenken im Zusammenhang mit BLE-Trackern zu adressieren, insbesondere angesichts der Unterschiede im Schutz zwischen iOS- und Android-Benutzern.

Ein Feature-basierter Prototyp wurde vorgeschlagen und drei Klassifikationsmodelle - SVM, Random Forest und Multi-Layer Perceptron - wurden evaluiert. Dabei erwies sich das Multi-Layer Perceptron Modell mit einer Genauigkeit von 94,5% auf Testdaten als das überlegene Modell. Weiterhin wurde dieses Modell auf unbekannte Geräte getestet, um seine Generalisierungsfähigkeit zu bewerten, und erreichte eine Genauigkeit von 88% mit einer binären Klassifikationszielsetzung - Tracker und Nicht-Tracker.

Nach Behebung eines identifizierten Fehlers in der ursprünglichen Anwendung wurde dieses Modell in die HomeScout-App integriert. Nun ist HomeScout in der Lage, zwischen Tracker- und Nicht-Tracker-Geräten zu unterscheiden. Zukünftige Arbeiten beinhalten die Verfeinerung des Prototyps, die Erweiterung der Vielfalt des Datensatzes und die Gewährleistung der Privatsphäre der Benutzer in öffentlichen Datensätzen.

0.4 Weitere Arbeiten

Der entwickelte Klassifikator ist ein Prototyp und bedarf weiterer Optimierungen, insbesondere in Bezug auf den Umfang des Datensatzes. Für eine verbesserte Klassifizierung von BLE-Geräten wäre es wünschenswert, einen größeren und vielseitigeren Datensatz zu erstellen, der weitere Gerätekategorien abdeckt.

MAC-Randomisierung ist weiterhin eine Herausforderung, und zukünftige Forschungen sollten darauf abzielen, Mechanismen zu entwickeln, um diese Geräte konsequent zu identifizieren. Weitere Features, wie die Ankunftszeit von BLE Advertisements, könnten ebenfalls in Betracht gezogen werden, um die Klassifikationsperformance zu steigern.

Beim Erstellen von öffentlichen Datensätzen sollte immer der Datenschutz gewährleistet sein, um die Privatsphäre der Nutzer zu schützen.

Abstract

In 2011, the introduction of Bluetooth Low Energy (BLE) marked a significant shift in wireless communication, paving the way for the Internet of Things (IoT) and the rise of location-based trackers. While devices like Apple's AirTag provide convenience, they pose security risks, notably the potential for malicious actors to track individuals unbeknownst to them. This work aims to address security concerns related to BLE trackers, especially considering the disparity between protections for iOS and Android users. The research focuses on creating an Android application, improving upon previous tools like HomeScout, which had limited classification capabilities. A feature based prototype was proposed and three classification models including SVM, Random Forest, and Multi-layer Perceptron were evaluated. The result was an effective classification method for BLE devices, with the Multi-Layer Perceptron model outperforming others with a 94.5% accuracy on test data. The model was further tested on unseen device to evaluate its generalization capability, which achieved a 88% of accuracy in with binary classification target, tracker and non-tracker. This model was integrated into the HomeScout app after resolving an identified bug in the original application. Eventually, Homescout is able to identify tracker and non-tracker device after integration. Future work entails refining the prototype, enhancing the dataset's diversity, and ensuring user privacy in public datasets.

Acknowledgments

I would like to express my sincere gratitude to Katharina and Prof.Stiller for giving me the valuable opportunity to conduct my master's thesis in the Communication Systems Research Group (CSG) at the Department of Informatics of the University of Zurich. Their guidance, support, and trust throughout this research journey have been instrumental in shaping the outcomes of my work.

Katharina's expertise, patience, and continuous feedback have been invaluable in refining my research methodology and enhancing the quality of my thesis. Her dedication to pushing the boundaries of knowledge and her unwavering commitment to my academic growth have been truly inspiring.

Last but not least, I would like to acknowledge the support and encouragement of my family and friends who have stood by me throughout this academic endeavor.

I am truly grateful to all those who have contributed to the successful completion of my master's thesis. Your support and guidance have been invaluable, and I am honored to have had the opportunity to work with such exceptional individuals at the CSG group.

Contents

Chapter 1

Introduction

In 2011, the introduction of Bluetooth Low Energy (BLE) technology marked a significant shift in the landscape of wireless communication. BLE was designed to be energy-efficient and optimized for short-range communication with small data payloads, making it an ideal technology for the Internet of Things (IoT) ecosystem. As a result, BLE has become an integral part of IoT and has facilitated the development of location-based trackers, which are increasingly popular. Its advantage in energy-saving perfectly suits the use case of location trackers, which require minimal battery drain to effectively track and report the location of items over extended periods. An increasing number of location trackers based on BLE network have been developed and introduced to the public. A study conducted by Circana [11] shows that sales for item trackers massively grew by nearly 63% in just the US during the months of January and February. The low price of these trackers also makes it accessible to the public and explain the reason why it's prevelant in the market.

One of the most well-known examples of these trackers is Apple's AirTag, which allows users to track and locate their belongings using their smartphones. It is a small, coin-shaped device that can be attached to an item using a keychain or adhesive, which is convenient for tracker users. For instance, users can attach an AirTag to their keys to help locate them in case they are misplaced or lost.

While these devices provide convenience to users, they also introduce security concerns that need to be addressed. For example, a malicious actor could potentially track the movements of an individual carrying an AirTag or similar device without their knowledge or consent. This raises privacy concerns, as the individual's location data could be used for malicious purposes. With the release of AirTag in 2011, concerns about these potential dangers have become increasingly prominent. More and more people have reported being stalked by AirTag [13]. Consumers are realizing these convenient IoT devices can also threaten their safety. Although device manufacturers and developers are implementing various security features in their products, the effectiveness of these measures is still being questioned. Apple implements a safety measure to Airtags which makes a beeping noise when an unregistered AirTag is detected moving with a user after 8 hours [16]. But people who intend to track other people can easily register an AirTag and track victims until they get home and then disable it. Besides, it's actually easy to muffle AirTag's alert having only a 60 decibel beep sound with some kind of cushions. Moreover, the current

state of protection varies between different smartphone platform. As of now, only iOS users are truly safeguarded, as Google and Apple have pledged to collaborate on ensuring Android users receive equivalent protection, though progress on this front has been slow. Additionally, users must consent to participate in the FindMy network, thus sharing their location with the network. However, if a user decides not to provide this consent due to privacy concerns, they will not receive alerts when an AirTag is tracking them. This contradiction further complicates the security issues of these tracking devices.

Thus, multiple mobile applications were developed, such as AirGuard [12] and HomeScout, to check for potentially dangerous devices. With the development of the HomeScout application in various previous projects, it is possible to detect and filter out personal trackers, including the AirTag, Tile, Samsung Galaxy SmartTag+, Chipolo One Spot, and the OpenHaystack tag. The application was designed to enable users to scan and detect personal tags at any location and throughout the day, thereby alleviating users' anxiety about being tracked. Furthermore, through the analysis of the scanned BLE devices and data contained in the transmitted packets, it has become apparent that BLE-based IoT devices can be similarly used to track objects or people. As such, it is important to expand the horizon of HomeScout beyond personal trackers towards a variety of BLE devices encountered in all facets of daily life.

Although HomeScout has shown its ability in filtering out personal trackers, it has limitations on the device identification ability. It can only identify trackers between AirTag, Chipolo ONE Spot, Galaxy SmartTag+ and Tile, which is achieved by analyzing the specific characteristic of each type individually. However, it's almost impossible to support all types of trackers on the market, especially given the vast number of existing and emerging trackers. Analyzing the specific characteristics of each tracker and incorporating them into the application's list of supported devices is a daunting task. Furthermore, another limitation of HomeScout is that it only supports the identification of trackers, leaving many other types of BLE devices remain unknown in the application, and consequently, users may still have concerns about those unknown devices.

Therefore, it is crucial to develop an application based on HomeScout that can passively scan and identify BLE devices and categorize them according to type or usage to enhance user awareness, enable control over personal data, and increase security awareness. This upgrade will help users have more awareness of the type of BLE devices around them and thus relieve users' concerns about safety significantly. The first step in achieving this is identifying and classifying IoT devices so that their purpose, communication, and location with respect to the user can be consistently recognized. As such, there have been a multitude of suggested approaches, such as behavioral fingerprinting [9], automated classification from network traffic streams [7], identification based on communication analysis [17], and deep packet inspection combined with keyword extraction for a rule-based classification approach. The last of which was demonstrated by IoTHunter [14], enabling the accurate distinction between different types of BLE devices, such as smart home devices, medical devices, or fitness trackers.

1.1 Motivation and Research Objectives

The motivation behind this research stems from the growing concerns related to security and privacy associated with trackers and other Bluetooth Low Energy (BLE) devices. The need to address these issues has driven our efforts to develop a method for effectively classifying various types of BLE devices, including location trackers.

One of the primary motivations is the disparity in the level of protection between iOS and Android users. Currently, iOS users enjoy enhanced security features against unwanted tracking, while Android users lack equivalent protection. To bridge this gap, our research focuses on creating an application specifically designed for the Android platform.

Previous efforts, such as HomeScout, have been focused on identification of personal trackers like Apple's AirTag. However, certain limitations exist in the classification capabilities of these existing solutions. Therefore, our research aims to overcome these limitations and develop an improved classification method that will be integrated into the HomeScout application.

The ultimate goal of our work is to enhance the functionality of HomeScout and provide Android users with a robust tool to detect and classify various BLE devices. By achieving this, we aim to alleviate user concerns about potential stalking or unauthorized tracking through the proactive identification and categorization of BLE devices encountered in daily life.

In summary, the main research objectives are as follows:

1. Develop a method for classifying different types of BLE devices, with a focus on location trackers and other IoT devices.
2. Integrate the developed classification algorithm into HomeScout, expanding its device identification capability beyond trackers.

1.2 Description of Work

The focus of this work is to explore link layer communication in the BLE network to recognize these devices and classify them into different categories.

The key contributions of this work include:

1. Examination of BLE link layer packets and exploration of their potential for classification.
2. Provision of a labeled dataset about BLE link layer packet, which provides the information about the characteristics of packet.
3. Design and implementation of machine learning models to classify BLE devices.
4. Integration of the developed algorithm into HomeScout, thereby enhancing its device identification capabilities.

1.3 Thesis Outline

This thesis is structured as follows: Chapter 2 presents related work in the area of BLE, offline finding networks, IoT devices identification and classification algorithm. Chapter 3 introduces the basics about BLE network, especially the link layer. And it introduces HomeScout, an Android app which our classification algorithm designed in this work will be integrated in. Chapter 4 describes analysis of collected BLE link layer packets, design and implementation of classification algorithm. Chapter 5 presents the results and evaluation of the conducted experiments about the classification algorithms. Chapter 6 discusses some findings of this work, including MAC randomization. Finally, Chapter 7 closes this work with the conclusion of this work and provides some insights of potential future work.

Chapter 2

Related Work

In this section, we provide a brief summary of the related work in two main categories: IoT device classification, BLE network analysis. These categories are the closely related to this work. After that, a research gap in the current academic community will be proposed.

2.1 IoT Device Classification

Given the limited existing research specifically dedicated to BLE device classification, insights and inspiration can be gained by looking at the broader field of IoT device classification. The knowledge and techniques developed for IoT devices can be adapted and applied to the context of BLE devices, enabling researchers to make meaningful progress in this area.

There have been a multitude of suggested approaches in IoT device classification, such as behavioral fingerprinting [9] [**?**], automated classification from network traffic characteristics [19], identification based on communication analysis [17], deep packet inspection combined with keyword extraction for a rule-based classification[14] and deep learning based approach[7].

[9] present a methodology for IoT device behavioral fingerprinting, where a device's behavior is determined using features extracted from its network traffic. These features are then used to train a machine learning model that can identify similar device types.

[19] introduces an unsupervised machine learning methodology for categorizing IoT devices based on traffic characteristics at the network level. The research addresses the challenge of device heterogeneity in IoT and the need for device classification for various applications. The study evaluates the performance of K-Means and BIRCH clustering algorithms on a real dataset.

In the study presented in [**?**], researchers introduced a fingerprinting technique using machine learning to identify Bluetooth wearables, specifically smartwatches. The primary feature utilized for the machine learning models was the distribution of inter-arrival-times of Bluetooth packets. Results demonstrated high precision and recall in identifying

wearables that operate on the Bluetooth classic protocol, indicating the effectiveness of the inter-arrival-time distribution as a distinguishing feature.

[17]introduces a device identification method that identifies device types and models based on general communication information. By calculating the similarity of features extracted from network packets, this method can identify various IoT devices without the need for specialized equipment.

IoTHunter[14] is a Deep Packet Inspection based IoT traffic classifier that extracts unique keywords, such as domain names and device names, to identify flows belonging to a particular device. The system automates the keyword extraction process by using the frequency of occurrence of words belonging to flows of different devices. To enhance performance, IoTHunter combines device-specific keywords with the MAC address of the device for subsequent flow labeling.

IoTHunter use the data from the same device for training and testing on a temporal basis and can thus only achieve intra-device identification and classify IoT devices already exist in the training dataset. [7] proposed a deep learning-based approach to classify unseen IoT devices by designing a feature extraction method and identifying invariant dependencies across devices. Their framework includes a LSTM-CNN cascade model that captures temporal correlations and effectively classifies IoT devices using real-world traffic data.

2.2 Bluetooth Low Energy Network

Although there is currently limited research on BLE device classifier, there are a lot of work focusing around broader field of BLE network, including environment detection using BLE advertising packets [18], analysis of Apple's Bluetooth Low Energy Continuity Protocols [15], tracking of BLE device with MAC address vulnerabilities [8] and BLE trackers detection[10] [12].

[18] is an AI-driven classifer that identify the category of environment. It scans BLE advertising packets and extracts features such as signal strength from them. The classification system of it is based on a two-layer dense neural network. It achieves an 84% of accuracy in differentiating between 7 categories of environment, including - home, office, shopping, transport, nature, street, and restaurant. The paper demonstrated the significant potential of leveraging BLE advertising packets for classification purpose.

[15] reverse-engineered various Continuity protocol message types and identified unencrypted data fields. They found that these messages are broadcast over BLE in response to various user actions, such as locking/unlocking the device, copying/pasting information, and making phone calls. The study also demonstrates that the content and format of Continuity messages can be used to fingerprint devices and profile users.

[8] introduces an address-carryover algorithm that exploits the asynchronous nature of payload and address changes to track devices beyond their address randomization cycles. The method extracts identifying tokens from the payload of advertising messages.

HomeScout[10] is an Android application used to detect stalking from BLE devices. It identifies malicious trackers that persistently appear in the user's vicinity and promptly alerts them. The parameters defining 'vicinity' and the frequency of 'persistent appearance' are adjustable, allowing users to customize the application according to their specific needs.

AirGuard[12] is an Android application developed to protect Android users from potential stalking threats associated with Apple's Find My network. It detects Find My enabled tracking devices through BLE background scans and provides users with timely notifications and guidance for locating the tracker.

2.3 Research Gap

Although there has been works focusing on IoT device classification and BLE network as described in section 2.1 and section 2.2. There remains a gap in the current literature, with no specific focus on classifying BLE devices-a step that could significantly alleviate user concerns about being tracked by devices like AirTags. To address this gap, this work introduces a novel BLE device classifier. For broader impact and seamless user experience, this classifier has been integrated into an existing Android application, HomeScout, creating a more comprehensive solution for mitigating privacy and security risks associated with BLE technology.

Chapter 3

Background

This chapter introduces the fundamental background for this thesis around BLE technology, HomeScout and three machine learning algorithms. Bluetooth Low Energy technology will first be introduced, which is the pivotal foundation for this research. After that, the HomeScout application will be described as the application on which the classifier is built. Additionally, its functionality and limitations will be elaborated on. Finally, we will introduce three popular machine learning algorithm, including Support Vector Machine, Random Forest and Multi-layer Perceptron. They will be leveraged to build the classifier of this work.

3.1 Bluetooth Low Energy

Bluetooth Low Energy (BLE), also known as Bluetooth Smart, is a wireless communication technology that enables low-power, short-range communication between devices. BLE was developed to address the growing demand for low-power, small-form-factor devices that can operate on a single coin-cell battery for extended periods of time. It is designed to be a low-power variant of the classic Bluetooth protocol, which is optimized for high-speed and high-throughput data transfer.

BLE operates in the 2.4 GHz ISM band and uses frequency-hopping spread spectrum (FHSS) to avoid interference from other wireless devices operating in the same band. It uses a master-slave architecture, with a central device (master) initiating and controlling communication with peripheral devices (slaves). BLE devices can operate in either a connected or advertising state, with the advertising state allowing devices to broadcast their presence and enable discovery by nearby devices.

BLE has several advantages over classic Bluetooth, including lower power consumption, smaller form factor, and lower cost. It is particularly well-suited for applications that require low power consumption, such as wearable devices, medical sensors, and Internet of Things (IoT) devices. BLE also has a relatively short range, which makes it ideal for use in environments where devices need to communicate over short distances, such as in a smart home or building automation system.

One of the key features of BLE is its ability to operate in a very low-power mode known as sleep mode. In this mode, the device consumes very little power and wakes up only when it receives a signal from a central device. This allows BLE devices to operate on a single coin-cell battery for extended periods of time, making them ideal for use in applications that require long battery life.

Another important feature of BLE is its support for GATT (Generic Attribute Profile) protocol, which enables devices to exchange data in a standardized format. GATT defines a set of services and characteristics that can be used to transmit data between BLE devices. This makes it easier for developers to create applications that work with a wide range of BLE devices.

Overall, BLE is a versatile and flexible communication technology that has enabled a wide range of low-power, small-form-factor devices to be developed. It is an important technology for the Internet of Things (IoT) and is expected to play a key role in the development of smart homes, smart cities, and other IoT applications.

3.1.1 Specifications

The Bluetooth specifications provide the foundation for developers to create devices that can communicate with each other. These specifications are managed by the Bluetooth Special Interest Group (SIG) and are regularly updated to keep pace with changing technology and market demands.

Core Specification

The Bluetooth Core Specification outlines the structure of the technology and its layers, describes its important features, and defines the formal procedures for key operations and protocols for devices to communicate at different layers of the stack. It's a comprehensive specification that guides developers in implementing a Bluetooth stack or its features. Essentially, it defines how Bluetooth technology works and the requirements for developers to follow when working with it.

Profile Specification

Profile specifications in Bluetooth Low Energy (BLE) define the behavior and functionality of devices in specific application domains. They standardize the way devices interact and communicate with each other to enable interoperability between different manufacturers and implementations.

Core Specification

The Bluetooth Core Specification outlines the structure of the technology and its layers, describes its important features, and defines the formal procedures for key operations and protocols for devices to communicate at different layers of the stack. It's a comprehensive specification that guides developers in implementing a Bluetooth stack or its features. Essentially, it defines how Bluetooth technology works and the requirements for developers to follow when working with it.

Service Specification

In Bluetooth Low Energy (BLE), state data on server devices is stored in formally defined data items called characteristics and descriptors. These characteristics and descriptors are organized and grouped within constructs called services. Services provide a context or framework for assigning meaning and defining behaviors to the characteristics and descriptors they contain.

A service specification outlines a specific service's details, including its characteristics and descriptors. It defines how the server device should behave and respond to various conditions and values of the state data. Essentially, a service specification represents one aspect of the behavior exhibited by the server device.

In simpler terms, services act as containers that hold characteristics and descriptors, which represent specific data and functionality of a device. The service specification provides guidelines on how the server device should operate within that service, defining its behavior and how it interacts with other devices.

Assigned Numbers

In the context of Bluetooth Low Energy (BLE), assigned numbers refer to unique identifiers that are assigned to different entities within the Bluetooth ecosystem. These assigned numbers are used to identify and differentiate between different types of devices, services, characteristics, and other entities that are part of the BLE protocol.

There are four types of assigned numbers used in BLE: company identifier, service UUID, characteristic UUID, and descriptor UUID. The company identifier is a 16-bit number that identifies a particular company or organization. The service UUID is a 128-bit number that identifies a specific service or feature that a device provides. The characteristic UUID is a 128-bit number that identifies a specific characteristic of a service, such as temperature or heart rate. The descriptor UUID is a 128-bit number that provides additional information about a characteristic, such as the units of measurement or the format of the data. The Bluetooth SIG manages the assignment of these numbers to ensure uniqueness and compatibility among BLE devices. The full list of assigned numbers can be found on Bluetooth SIG's official website[1].

3.1.2 Discovery Process

In the Bluetooth Low Energy (BLE) Link Layer discovery process, devices utilize advertising channels to find each other. One device operates in advertising mode, broadcasting its presence and information, while another device operates in scanning mode, actively listening and searching for advertising packets.

Advertising

BLE Advertising Broadcast, commonly known as advertising, is a communication mode in Bluetooth Low Energy (BLE) that operates without establishing a connection. It serves two main purposes: transferring data and indicating the availability of a peripheral device for connection.

Advertising packets are designed to be received by any scanning device within range. Therefore, advertising allows for the simultaneous transmission of data to multiple scanning devices in a one-to-many topology. However, there is also a special form called directed advertising, which enables the one-to-one communication of data from an advertising device to a specific scanning device identified by its Bluetooth device address.

Advertising is primarily unidirectional, enabling data transmission from the advertising device to scanning devices. However, scanning devices can reply to advertising packets with Protocol Data Units (PDUs) to request additional information or initiate a connection.

It's important to note that advertising is considered an unreliable transport method since the receivers send no acknowledgments. This means that there is no guarantee of successful packet delivery or reception.

The Bluetooth Core Specification defines two categories of advertising procedures: legacy advertising and extended advertising. Legacy advertising will be mainly discussed in this thesis.

Scanning

There are two types of scanning: passive scanning and active scanning. In passive scanning, the scanner device solely listens for advertising packets without actively interacting with the advertiser. The advertiser, on the other hand, is unaware of whether its advertising packets have been received by any scanning device. Active scanning is typically employed when a potential central device requires additional information beyond what is provided in an ADV_IND (advertising indication) packet before deciding to establish a connection. During an advertising interval, the scanner device initiates the active scanning process by sending a SCAN_REQ (scan request) packet to the advertising device. In response, the advertiser sends a SCAN_RSP (scan response) packet containing supplementary information to the scanner. This exchange allows the scanner to gather more details about the advertiser, aiding in the decision-making process of whether to proceed with a connection.

3.1.3 Protocol Stack

The Bluetooth Low Energy (BLE) protocol stack is a set of smaller protocols that cooperate to provide the overall functionality of the BLE protocol. It consists of three primary subsystems, namely the application, host, and controller blocks.

The application part acts as an interface between the user and the Bluetooth Low Energy protocol. It integrates the entire BLE protocol's functionality into a package that is easily accessible by the user. The host subsystem is typically a software stack that includes the uppermost layers of the BLE stack and profiles. The controller subsystem comprises the lower layers of the BLE stack. It includes the physical layer(PHY) and link layer(LL) of BLE. An overview of layers included in each part can be seen in Figure 1.

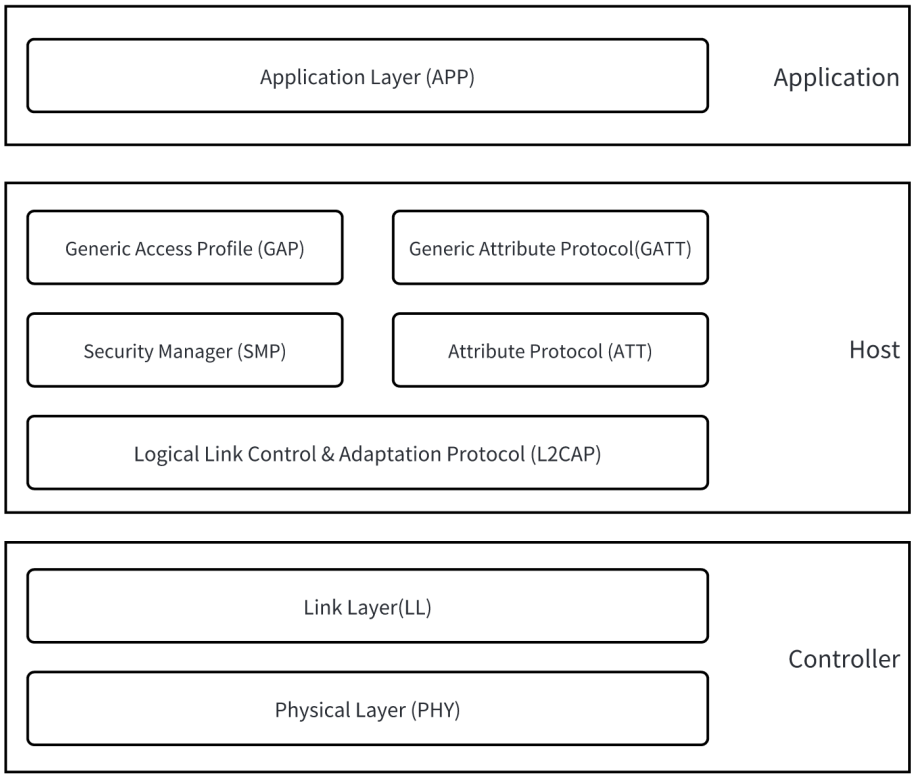


Figure 1: The Bluetooth Low Energy Protocol stack.

3.1.4 Physical Layer

The physical layer of Bluetooth Low Energy (BLE) is responsible for establishing and maintaining the physical communication link between BLE devices. It defines the modulation scheme, frequency hopping pattern, and transmission power levels used for data transmission.

The BLE physical layer operates in the 2.4 GHz ISM band, which is divided into 40 channels, each with a 2 MHz bandwidth. To minimize interference and maximize the ro-

bustness of the communication, BLE employs frequency hopping spread spectrum (FHSS) technique. This means that the BLE devices switch between these channels at a rapid rate, typically hopping 1600 times per second.

BLE uses a Gaussian frequency shift keying (GFSK) modulation scheme to encode data for transmission. This modulation scheme helps in achieving a good trade-off between power efficiency and spectral efficiency, making it suitable for low-power and low-complexity applications. GFSK modulation allows the transmitted signal to occupy a wider bandwidth compared to simple on-off keying, which helps combat multipath fading and interference.

The transmission power levels in BLE are adjustable, allowing devices to optimize the communication range and power consumption based on their specific requirements. The power levels range from -20 dBm to +20 dBm, with increments of 4 dBm. Lower power levels result in reduced communication range but also lower power consumption.

Furthermore, the BLE physical layer defines two types of channels: advertising channels and data channels. Advertising channels are used for device discovery and initial connection establishment, while data channels transmit actual data between connected devices. Advertising channels utilize three of the 40 available channels (channel indices 37, 38, and 39), while data channels can use any of the remaining 37 channels.

In summary, the physical layer of BLE encompasses the use of the 2.4 GHz ISM band with 40 channels, frequency hopping spread spectrum (FHSS) technique, Gaussian frequency shift keying (GFSK) modulation scheme, adjustable transmission power levels, and the differentiation between advertising and data channels. These specifications enable BLE devices to establish reliable and low-power communication links.

3.1.5 Link Layer

The Link Layer of Bluetooth Low Energy (BLE) is a fundamental component of the BLE protocol stack. It is responsible for establishing and maintaining reliable communication links between BLE devices. It specifies the format of several types of packets that are transmitted in the context of BLE. The Link Layer handles critical functions such as device discovery, connection establishment, data packet transmission, and error handling. It supports advertising mechanisms to broadcast device information and allows devices to discover each other. It also enables the establishment of different types of connections, including master-slave and peer-to-peer connections.

BLE Packets

BLE v5.1 specifications defines two packet types. The first is used by the uncoded PHYs , LE 1M and LE 2M, as shown in Figure 3. The second type is used by the LE Coded PHY as shown in Figure **??**. Given that most devices used the uncoded PHYs, the BLE packets format, and relevant knowledge discussed will be focused on it in the following discussion.

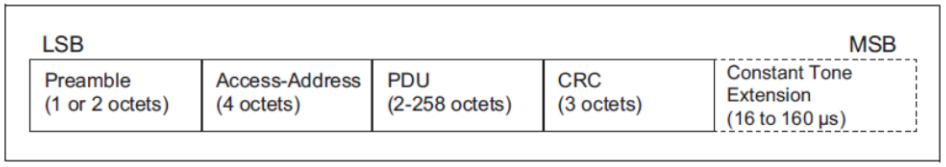


Figure 2: Link layer packet format for the LE uncoded PHYs. Source [2].

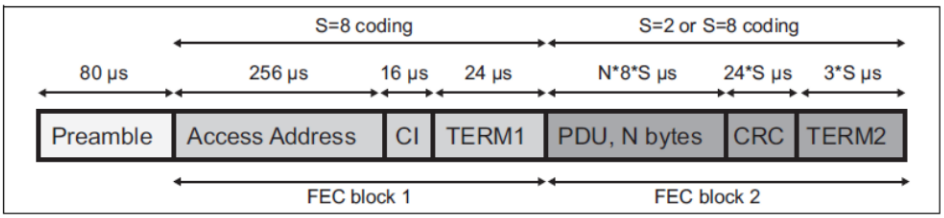


Figure 3: Link layer packet format for the LE Coded PHY. Source [2].

For ease of use, BLE utilizes a unified packet format for both advertising and data transmissions. It consists of four main components: Preamble, Access Address, Protocol Data Unit(PDU), and Cyclic Redundancy Check(CRC). The preamble is a one-octet field that acts as a synchronization sequence. It helps the receiver detect the start of the packet and synchronize its clock. Access address is a four-octet field that provides a unique identifier for the Bluetooth packet. It helps differentiate between different Bluetooth networks operating in the same area. PDU carries the actual data or control information within the packet. Its length can vary from 2 to 257 octets, depending on the type of packet and the information it carries. CRC is a three-octet field that contains error-checking information. It is used to detect and correct transmission errors, ensuring the integrity of the data.

PDU

Depending on whether packets are transmitted on an advertising channel or data channel, the format of PDU has two variants. The format of Advertising Channel PDU is shown in Figure 4 as the format of Data Channel PDU is shown in Figure 5. As this work focuses mostly on advertising packets, the PDU discussed in the following will also focus on Advertising Channel PDU.

Advertising Channel PDUs serve two primary purposes: Broadcast data for applications that do not require a full connection and discover slaves and connect to them [3]. The

Figure 4: Advertising Channel PDU format. Source [2].

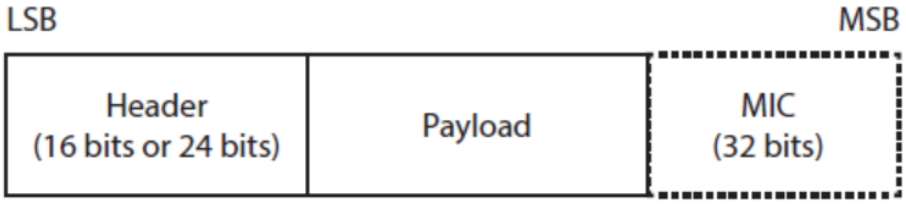


Figure 5: Data Channel PDU format. Source [2].

Advertising PDU packet contains a 16-bit header and a variable-size payload. The header contains six segments, as displayed in Figure 6.

PDU Type in the header indicates the type of PDU being used. RFU is Reserved for future use. ChSel is 1 if LE Channel Selection Algorithm is supported. TXAdd is 0 if the transmitter address is public, 1 if random. RXAdd is 0 if the target's address is public, 1 if random. The length segment indicates payload length. PDU type can generally be divided into Legacy Advertising and Extended Advertising. The details of the Legacy Advertising PDU type are shown in Figure 7. Extended Advertising PDU type will not be discussed as devices in this work don't use extended advertising.

ADV_IND packet

ADV_IND packet is used when a broadcaster wants to broadcast its presence and provide basic information to unspecified devices, known as undirected advertising. The difference between it and ADV_NONCONN_IND is ADV_NONCONN_IND doesn't accept the connection, which is commonly used in Beacon applications. For ADV_IND packets, the peripheral device requests connection to any central device. It serves as a means for the advertising device to actively notify potential scanning devices of its existence and availability. It contains essential information such as the device's identity, capabilities, and available services. This packet is designed to be received by any scanning device within range.

The payload of ADV_IND packet consists of the advertiser device address and advertisement data as shown in Figure 8. The advertisement data is specified by the manufacturer of the peripheral device to deliver information about itself. It consists of several Advertisement Data Structures having the following format: AD Length, AD type, and AD data. This follows the well-known LTV (length-type-value) format. The structure of advertisement data is shown in Figure 9. AD length indicates the length of AD type and AD data. AD type indicates the type of AD data, which is specified in the assigned number specifications of Bluetooth [1]. AD data contains the data that the manufacturer

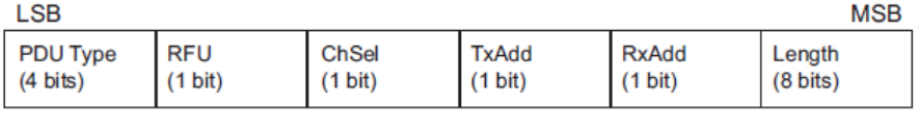


Figure 6: Advertising PDU header. Source [2].

PDU Name	Description	Channels	PHY(s)	Transmitted By	Scannable	Connectable
ADV_IND	Undirected advertising	primary	LE 1M	Peripheral	Y	Y
ADV_DIRECT_IND	Directed advertising	primary	LE 1M	Peripheral	N	Y
ADV_NONCONN_IND	Undirected, non-connectable, non-scannable advertising	primary	LE 1M	Peripheral	N	N
ADV_SCAN_IND	Undirected, scannable advertising	primary	LE 1M	Peripheral	Y	N
SCAN_REQ	Scan request	primary	LE 1M	Central	N/A	N/A
SCAN_RSP	Scan response	primary	LE 1M	Peripheral	N/A	N/A
CONNECT_IND	Connect request	primary	LE 1M	Central	N/A	N/A

Figure 7: Advertising PDU type. Source [5]

decides to deliver, which typically contains information like device name, manufacturer information, and other related information.

SCAN_RSP packet

In Bluetooth Low Energy (BLE), the Scan Response (SCAN_RSP) packet is a type of packet used in response to a scan request from a scanning device. The SCAN_RSP packet is sent by a peripheral device and provides additional information beyond what is available in the advertising packet.

The SCAN_RSP packet typically follows the scan request initiated by the scanning device, in which the SCAN_REQ is sent. It contains data fields that are structured to convey specific information such as device identification, service availability, or any other relevant data that the advertising device wishes to communicate.

The structure of SCAN_RSP packet is typically similar to ADV_IND, which is introduced in detail in the last section.

State Machine

The Link Layer of Bluetooth Low Energy (BLE) employs a state machine to manage its operations and ensure efficient and reliable communication between devices. It consists

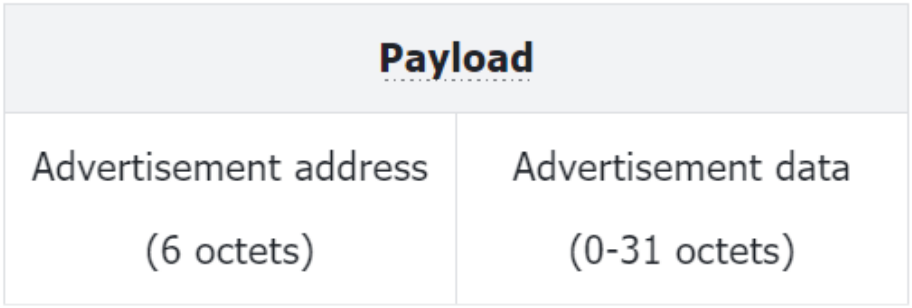


Figure 8: ADV_IND payload.

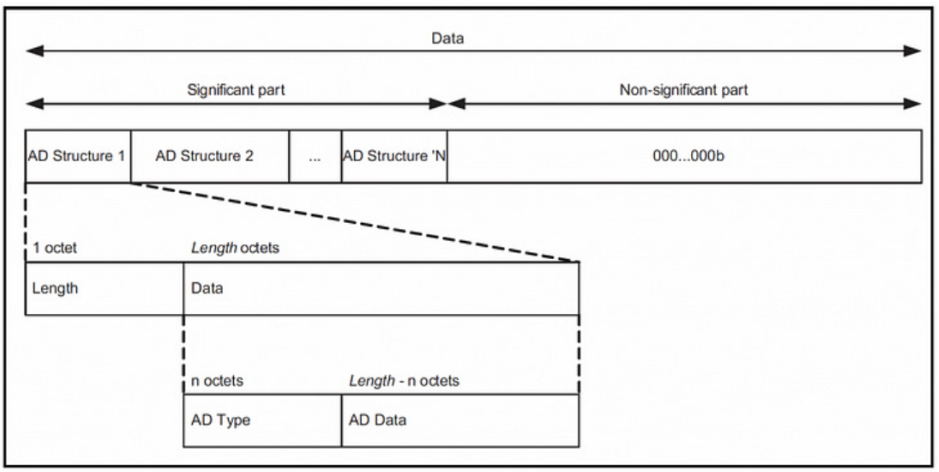


Figure 9: ADV_IND payload AD structure. Source [5].

State	Description
Standby	Device neither transmits or receives packets.
Initiating	Responds to advertising packets from a particular device to request a connection.
Advertising	Transmits advertising packets and potentially processes packets sent in response to advertising packets by other devices.
Connection	In a connection with another device.
Scanning	Listening for advertising packets from other devices.
Isochronous Broadcast	Broadcasts *isochronous* data packets.
Synchronization	Listens for periodic advertising belonging to a specific *advertising train* transmitted by a particular device.

Figure 10: The link layer state machine. Source [5].

of seven states, which is shown in Figure 10.

At a high level, the Link Layer state machine has three main states: Advertising, Initiating, and Connection. In Advertising State, the device actively broadcasts advertising packets to nearby devices. It waits for connection requests and handles events such as receiving connection requests or timing out without receiving any requests. When a connection request is received, the state machine transitions to the Initiating state. Upon transitioning to initiating state, the device sends a connection request to the desired advertising device. It awaits a connection response and handles events such as receiving a connection response, timing out, or encountering an error. If a connection response is received and validated, the state machine transitions to the Connected state. Connected State represents an established connection between two BLE devices. It involves bidirectional data exchange, error handling, and various connection-related events. The state machine handles events such as data transmission, disconnection requests, link-layer encryption, and periodic supervision timeout. Based on these events, the state machine can transit to different states as necessary.

3.2 HomeScout

HomeScout is an Android application designed to address safety concerns associated with BLE trackers [10]. The main functionality of this Android app is to perform scanning, identification, and classification of BLE trackers as either malicious or non-malicious. However, the tracker identification is implemented based on individual analysis, which doesn't provide a holistic classification. Thus, the overall structure, functionality, and limitations of HomeScout will be introduced in this section.

3.2.1 App Workflow

The main functionality is divided into three services, which are depicted conceptually in Figure 12. Firstly, tracking protection needs to be enabled. Afterward, the app evaluates whether the user is moving or stationary. Once the user starts moving, BLE advertisements are scanned and saved in a database. Simultaneously, the tracking algorithm attempts to detect any malicious BLE device among all the scanned BLE devices or advertisements.

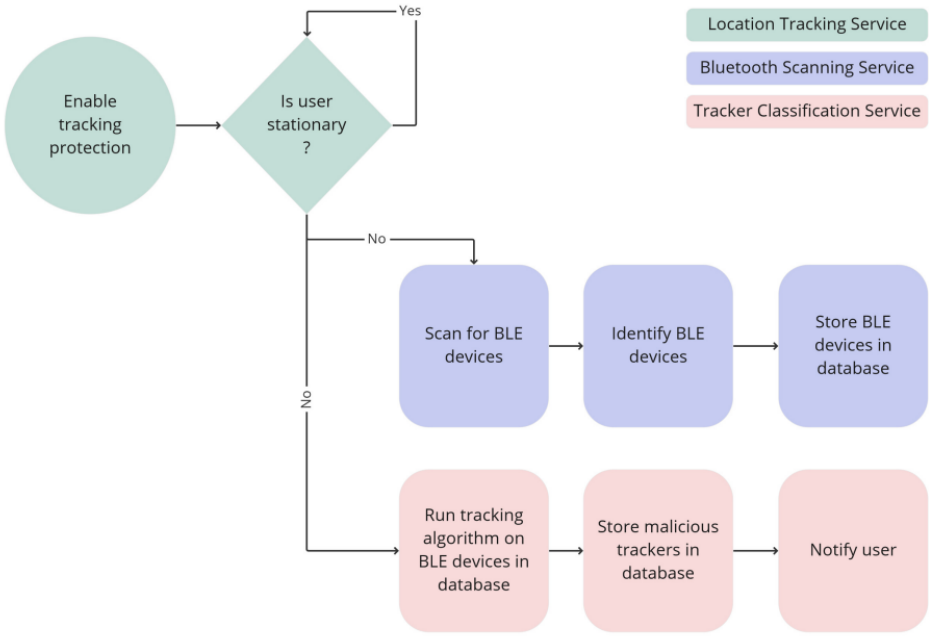


Figure 11: Sketch of the app workflow. Source [10].

3.2.2 App Architecture

The general architectural components of the app are the User Interface (UI), the data layer and the services. Figure 12 shows the general architecture of the app.

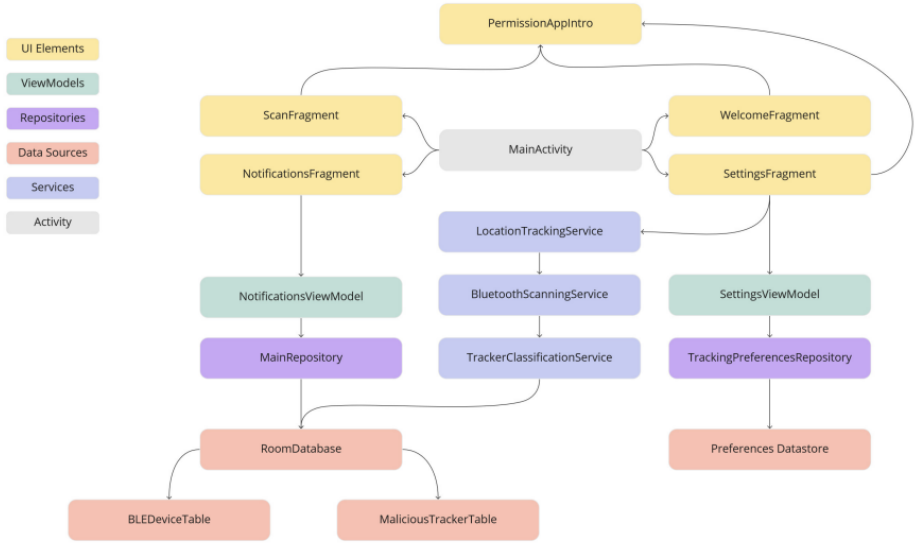


Figure 12: App architecture of HomeScout. Source [10].

3.2.3 User Interface

The user interface is comprised of the app intro and UI layer itself. The app intro consists of four screens to guide users to request necessary permissions.

App Intro

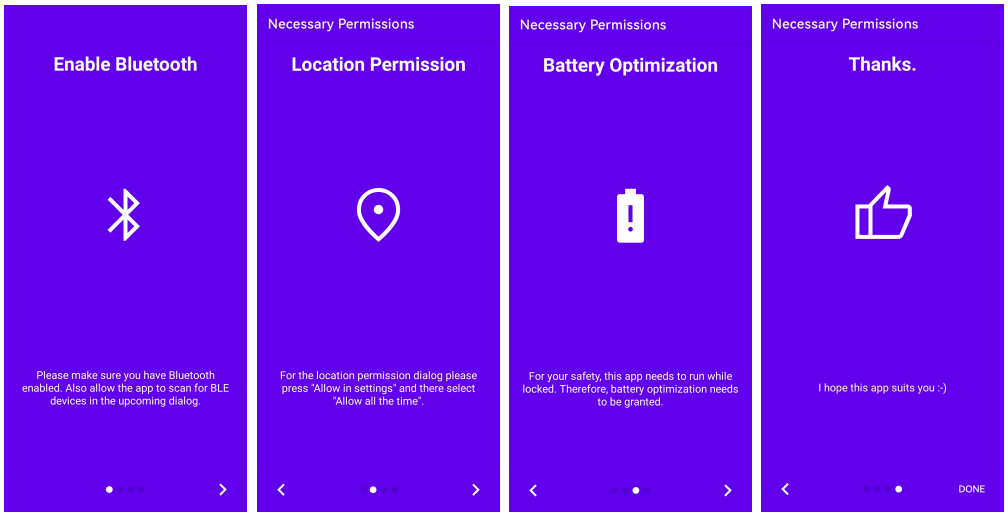
The app consists of four screens. The first screen informs the user about the Bluetooth permission. The second screen explains the location permissions required for the app to access background locations. The third screen serves as a placeholder for requesting the user to ignore battery optimization. The last screen expresses gratitude to the user for granting the requested permissions. Figure 14 displays screenshots of these four screens.

UI Layer

The UI layer also consists of four pages, Welcome, Notifications, Settings, and Scan fragments. Figure 14 shows screenshots of those four fragments including the bottom navigation bar.

3.2.4 Services

The tracking protection functionality of HomeScout is divided into three separate services. The first service tracks the user's location, while the second service scans for BLE devices if necessary. The third service applies a classification algorithm to the scanned BLE devices



(a) (b) (c) (d)

Figure 13: Screenshots of four pages in app intro. a) asks for Bluetooth, b) for location, and c) for battery optimization permissions. Finally, d) thanks the user for granting the permissions.

and alerts the user when a malicious tracker is detected. Each of the three services will be introduced in detail in this section.

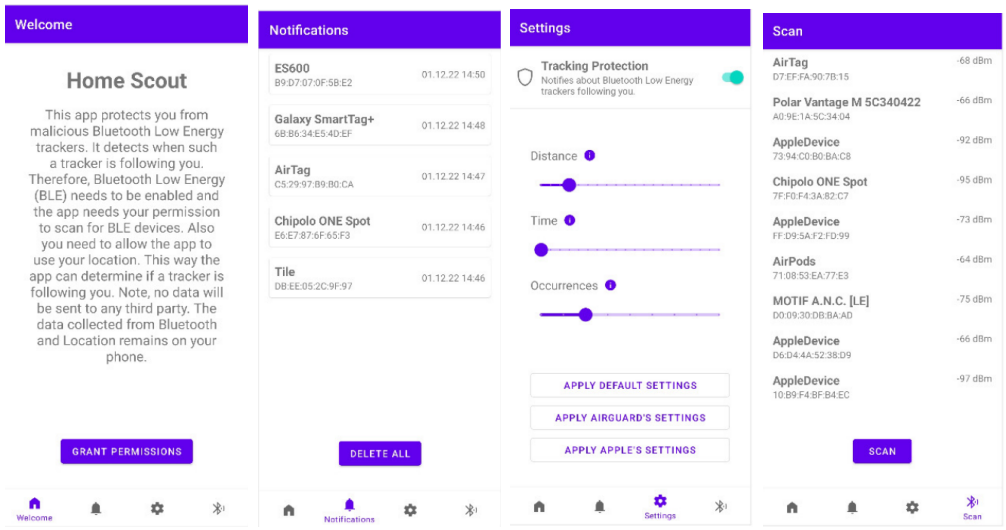
The main focus of this work is to enhance and modify the third service, namely classification algorithm, in HomeScout. The current algorithm is not comprehensive as it is derived from individual analysis. In this section, the third service of HomeScout will be introduced in details so as to understand the limits of it.

Location Tracking

The LocationTrackingService determines user mobility, initiating the BluetoothScanningService and TrackerClassificationService only when the user is in motion. This strategy helps conserve battery life and targets potential threats during user movement. The user's mobility status is determined by creating a "tail" of previous positions using a ring buffer data structure. If the user's travelled distance, calculated based on this tail, exceeds 50 meters, it is assumed the user is in motion.

Bluetooth Scanning

The BluetoothScanningService controls the scanning process for BLE devices in the app, optimizing battery use by managing scan periods and intervals. It initiates scanning through the startBleScan function and sets up handlers to define a 12-second scan period and an 18-second scan interval. This ensures signals from all BLE devices are captured, with the scanning process repeating twice per minute until the user is stationary or disables tracking protection.



(a) (b) (c) (d)

Figure 14: Screenshots of four UI pages in HomeScout. a) Welcome fragment b) the Notifications, c) the Settings page and d) the Scan fragment. Source [10].

This scanning process yields a ScanResult, an object defined in Android APIs. It describes information about a detected device and consequently BLE link layer packets can be retrieved. Our classification algorithm is intergrated into HomeScout from this point. It mainly makes use of this service to get BLE link layer packets, which serves as the data source of classification process

Tracker Classification

The TrackerClassificationService in HomeScout classifies BLE devices as either malicious trackers or non-threatening entities, operating every 30 seconds for accuracy and battery efficiency.

The service organizes scanned BLE devices in a hashmap, sorted by MAC addresses and scan time. The classification algorithm evaluates maliciousness using three user-set parameters: occurrences, timeInMin, and distance.

First, it checks occurrences, requiring at least two detections for further assessment. Devices with insufficient occurrences are deemed non-malicious.

Next, it examines the time span between the earliest and latest scans, comparing this against the timeInMin parameter. Devices that exceed the allowed time are not labeled as malicious.

Finally, it assesses the distance traveled by the user between scan points. Devices that surpass the distance threshold are also deemed non-malicious.

However, the system has limitations. The dependency on user-set parameters may lead to misclassification if set inaccurately. Also, this classification service depends on the

successful identification of trackers, which is illustrated and demonstrated to be unreliable in next section due to its limited generalization ability.

3.2.5 Tracker Identification

In Bluetooth scanning segment of HomeScout, trackers including AirTag, Chipolo ONE Spot, Galaxy SmartTag+ and Tile can be classified from their BLE advertisement with individual analysis. It doesn't provide a comprehensive classification method to identify trackers. The classification is derived from analyzing the characteristic of the advertisement packet of the four trackers individually. The pattern of each trackers is investigated and then used to identify. The pattern analysis of four trackers will be introduced in this section.

AirTag

The identification of AirTag in HomeScout is based on results of reverse engineering of BLE advertisements of Apple devices by Heinrich et al [12]. The interpretation and processing of this manufacturer-specific data are necessary to eventually identify an AirTag as such. The reverse engineering result of Apple devices' advertisement packet format is shown in Figure 15. And from reading status byte of the advertisement packet, AirTag can be identified. By applying and() operation with 0x30 and a right shift with 4 bits, the third and fourth bits of the status byte can be retrieved. If the third and fourth bits equal 0b01, the device can be identified as an AirTag.

Bytes	Content
0-5	BLE address
6	Payload length in bytes (30)
7	Advertisement type (0xFF for manufacturer-specific data)
8-9	Company ID (0x0004C)
10	Offline Finding type (0x12)
11	Data length in bytes (25)
12	Status (e.g. battery level)
13-34	Public key bytes
35	Public key bits
36	Hints (0x00 on iOS reports)

Figure 15: Apple's advertisement packet format. Source [10].

Chipolo One Spot

Chipolo One Spot works as part on Apple's Find My Network, which means it can be identified using the same method as AirTag. The only difference is the third and fourth bits of it equals 0b10.

Galaxy SmartTag+

The identification of Galaxy SmartTag+ is managed by reading the local name in the advertisement packet. Galaxy SmartTag+ expose its name in the payload, which is straightforward for identification.

Tile

Tile is identified by UUID in the advertisement packet. A UUID is a universally unique identifier. If the UUID found in advertisement packet equals 0000FEED-0000-1000-8000-00805F9B34FB, the device will be classifed as a Tile tracker. However, the accurary of this method is in doubt, as there is no guarantee that the UUID is assigned to a Tile tracker uniquely.

3.3 Support Vector Machines

Support Vector Machines (SVMs) are a category of high-performance supervised learning models used in machine learning and statistics for both classification and regression tasks.

3.3.1 Basic Concept and Mechanism

The core principle of an SVM is to construct a hyperplane, or a set of hyperplanes, in a high or infinite dimensional space that can be utilized for classification, regression, or other tasks. In the realm of classification, the key intuition behind SVM is to find the optimal hyperplane that separates the data into two classes, while maximizing the margin between the classes.

In this scenario, a hyperplane is a decision boundary separating the feature space into two half spaces. Each half space corresponds to a particular class label. In a two-dimensional space, a hyperplane is simply a line, while in a three-dimensional space, it becomes a plane, and so on.

The margin in an SVM context is defined as the distance between the nearest data point from each class and the separating hyperplane. The data points that are closest to the decision boundary are known as support vectors. SVMs strive to maximize this margin, hence creating the largest possible distance between the decision boundary and the nearest instances from both classes.

Maximizing the margin is beneficial as it contributes to better model generalization, reducing the risk of overfitting. Models with larger margins are more robust to noise and have a better predictive capability for unseen data.

3.3.2 Kernel Functions

Kernel functions play an integral role in SVM's capacity to classify non-linearly separable data by transforming the input space into a higher-dimensional feature space. This transformation allows us to draw a linear separating hyperplane in this new space, thus enabling SVM to handle non-linearly separable problems.

There are several types of kernel functions that are commonly used in SVMs:

- **Linear Kernel:** The simplest form of kernel function is the linear kernel, given by the dot product of two input vectors. The resultant SVM is a linear classifier, operating in the original feature space.

- **Polynomial Kernel:** Polynomial kernels, expressed as $(\gamma\langle x, x'\rangle + r)^d$, allow SVM to classify data that is separable by a polynomial decision boundary. Here, $\langle x, x'\rangle$ denotes the dot product, d is the degree of the polynomial, and γ and r are kernel parameters.

- **Radial Basis Function (RBF) or Gaussian Kernel:** Arguably the most widely used kernel function, the RBF kernel allows the construction of a decision boundary that is a smooth curve in the original feature space. It is expressed as $\exp(-\gamma|x - x'|^2)$, with γ determining the spread of the Gaussian.

- **Sigmoid Kernel:** The Sigmoid kernel, given as $\tanh(\gamma\langle x, x'\rangle + r)$, is similar to the hyperbolic tangent activation function used in neural networks and provides a decision boundary that resembles a step function in the original feature space.

Choosing the right kernel function for an SVM is crucial and typically depends on the nature of the data and the problem at hand. A good choice of kernel can greatly enhance the performance of the SVM, enabling it to handle complex, high-dimensional data, and classify it accurately.

However, using a kernel function, especially a non-linear one, increases the risk of overfitting, especially with a high-dimensional feature space. To mitigate this risk, parameters of the kernel function are often tuned using techniques such as cross-validation to balance the trade-off between model complexity and overfitting.

3.3.3 Advantages and Disadvantages

SVMs come with a set of unique benefits and challenges that dictate their application in certain scenarios:

Advantages

- **Effective in high dimensional spaces:** SVMs are especially effective when the number of dimensions is greater than the number of samples.
- **Versatility:** Different kernel functions can be specified for the decision function, offering flexibility to model various types of data.
- **Maximal Margin:** SVMs aim to maximize the margin around the separating hyperplane, which tends to result in robust classification.
- **Overfitting control:** SVMs provide good generalization performance, which is controlled by the hyperparameters C (regularization parameter) and parameters of the kernel function.

Disadvantages

- **Need for careful preprocessing:** SVMs require careful preprocessing of the data and tuning of the parameters. This includes scaling of the data, selection of a suitable kernel, and tuning of the hyperparameters.
- **Poor performance with large datasets:** SVMs do not perform well when the dataset is very large or when there are a lot of noise and overlapping classes.
- **No probability estimation:** SVMs do not directly provide probability estimates. These are calculated using expensive five-fold cross-validation.

3.4 Random Forest

Random Forest is a powerful, ensemble-based machine learning algorithm that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes output by individual trees for classification tasks, or mean prediction of the individual trees for regression tasks.

3.4.1 Decision Tree

A decision tree, a core component of a Random Forest, is a model presented in a flowchart-like structure. It is composed of nodes, branches, and leaf nodes, and its purpose is to divide the dataset into groups or classes based on a variety of conditions.

Nodes are points of data division. The root node, positioned at the top, partitions the data following a specific criterion. This division persists through what are termed internal nodes, which continue the process until a leaf node is reached. Each individual node assesses a specific attribute.

Branches serve to portray the results of a test at a node, and they provide a connection between nodes, indicating the flow from one node to the next.

Leaf nodes, also referred to as terminal nodes, represent the final outcomes or decisions. Once a data sample reaches a leaf node, it is assigned a class label.

The decisions made by decision trees are straightforward and easy to understand, which contributes to their popularity. Their decisions can be visualized and interpreted logically, which is a significant advantage when model interpretability is essential.

A crucial element in decision tree creation is the selection of which attribute to test at each node. Several metrics, such as Gini impurity and information gain, can aid in determining the best attribute.

While a single decision tree may be susceptible to overfitting, especially if it is allowed to grow too complex or deep, the Random Forest algorithm mitigates this problem by creating an ensemble of different decision trees and using their aggregate predictions. Each tree in the ensemble is trained on a different subset of the training data, which contributes to the diversity of the trees and ultimately leads to a more robust and generalizable model.

3.4.2 Basic Concept and Mechanism

The Random Forest algorithm is built on the principle of ensemble learning, where multiple learning models (in this case, decision trees) are developed independently and their predictions are combined to produce the final output.

Bagging and Bootstrapping

Random Forest uses a technique called bagging (bootstrap aggregating), which is crucial to its operation. Bagging involves creating multiple subsets of the original dataset, with replacement, where each subset is the same size as the original set. These subsets, called bootstrap samples, are then used to train individual decision trees. This means that each decision tree is trained on a different dataset, adding variety to the ensemble and improving the model's robustness. Moreover, bootstrapping reduces the variance of the prediction, thereby helping to prevent overfitting.

Decision Trees and Feature Randomness

Random Forest constructs a multitude of decision trees, where each tree is built by splitting on a series of nodes. At each node, the split is determined by the feature that provides the best separation of the data, according to a certain criterion such as Gini impurity or information gain. However, in a Random Forest, rather than considering all features, a random subset of features is selected at each node, further promoting diversity within the ensemble.

Each decision tree is grown to the maximum depth, meaning that no pruning is done. While this could lead to overfitting in a single decision tree, it is not a concern in a Random Forest because the final prediction is the aggregate of all the trees, which helps to balance out any overfitting that might occur in individual trees.

Prediction Aggregation

Once the decision trees have been trained, predictions are made by feeding the new data through each tree in the forest. Each tree makes a prediction independently, and the final output is determined by aggregating these predictions. For classification problems, the most common class (mode) predicted by the trees is selected, whereas, for regression problems, the average (mean) prediction of the trees is taken. This approach leverages the wisdom of the crowd, reducing the impact of individual tree errors and leading to more accurate and robust predictions.

3.5 Multi-layer Perceptron

A Multi-layer Perceptron (MLP) is a class of feedforward artificial neural network that is widely used in supervised learning scenarios, such as classification and regression tasks. An MLP consists of at least three layers of nodes: an input layer, a hidden layer, and an output layer. Each node, also known as a neuron, within a layer is connected to every neuron in the subsequent layer, forming a fully connected network.

3.5.1 Basic Concept and Mechanism

The foundation of an MLP is the perceptron, a binary classifier that maps a set of input features to an output via a set of weights. In an MLP, these perceptrons are organized in layers. The input layer corresponds to the features in the dataset, the hidden layer performs computations and transformations on these inputs, and the output layer generates the final output of the network.

Each neuron in the MLP takes the weighted sum of its inputs, adds a bias term, and then applies a non-linear activation function. The weights and biases are parameters of the model that are learned during training. The activation function introduces non-linearity into the model, which allows the MLP to model more complex relationships between the inputs and the output.

Backpropagation is the key algorithm for training an MLP. During backpropagation, the model makes a prediction, calculates the error (difference between the prediction and the actual value), and then propagates this error back through the network, adjusting the weights and biases along the way to minimize this error. This process is repeated multiple times on batches of data from the dataset until the model's predictions are satisfactory.

3.5.2 Activation Functions

Activation functions play a critical role in neural networks, as they introduce non-linearity into the model, which enables the network to capture more complex patterns in the data. There are several activation functions commonly used in MLPs:

The sigmoid function maps its input to a value between 0 and 1, which can be used to represent probabilities. The hyperbolic tangent function (tanh) also maps its input to a value between -1 and 1, providing a zero-centered output which can help the model learn more effectively. The Rectified Linear Unit (ReLU) function maps its input to a value that is either 0 (for negative inputs) or the input value itself (for positive inputs). ReLU is often used in the hidden layers of neural networks due to its computational efficiency.

3.5.3 Advantages and Disadvantages

Advantages

- **Capability to Model Non-linear Relationships:** The key advantage of MLPs is their ability to model non-linear relationships. This is achieved through the use of multiple layers of neurons and non-linear activation functions. This feature makes MLPs highly flexible and powerful, able to model complex functions and patterns in data that linear models might miss.

- **Availability of Learning Algorithms:** With the backpropagation algorithm and the power of modern computing, MLPs can be efficiently trained on large datasets. Furthermore, various optimization techniques, such as gradient descent and its variants, can be used to speed up the learning process.

- **Scalability:** MLPs can easily be scaled up by adding more hidden layers or neurons per layer to handle larger and more complex datasets.

Disadvantages

- **Risk of Overfitting:** Due to their flexibility and capacity to fit complex functions, MLPs are prone to overfitting, especially when dealing with high-dimensional data or when the network has too many layers or neurons. Overfitting occurs when the model learns the training data too well, including its noise and outliers, which negatively impacts the model's generalization ability.

- **Requirement for Large Datasets:** MLPs usually require large amounts of training data to perform well and avoid overfitting. This is due to the large number of parameters (weights and biases) that need to be learned. Insufficient training data can lead to a poorly performing model.

- **Opaque Decision Making:** MLPs, like many neural networks, suffer from being "black box" models, meaning their decision-making process is not readily interpretable. This lack of transparency can be a drawback in applications where interpretability is important.
- **Sensitivity to Input Scaling:** MLPs are sensitive to the scaling of input features. Features need to be properly normalized or standardized before being fed into the network, or else features with larger scales may dominate the learning process.

Chapter 4

Design and Implementation

In this chapter, the design process and implementation details of a BLE device classifier will be introduced. The classifier developed in this work will be incorporated into a pre-existing Android application known as HomeSocut. However, the development and testing of algorithms on the Android platform pose certain challenges. Therefore, to circumvent these hurdles, data capture, algorithm design, development, and testing are initially conducted on the Windows 10 platform. The transfer of the algorithm to the Android platform is a straightforward process. The major complexity arises from the differences in the BLE packets accessible via Android APIs versus the data captured on a PC. The analysis of this problem is thoroughly explained in Section 4.1.

4.1 Data Acquisition

Considering the abundant information available in the link layer of BLE networks, as discussed in Section 3.1, this work primarily relies on BLE link layer packets as a valuable source of data. The specific characteristics of BLE broadcast packets, such as their frequent transmission, the transmission of data in plaintext, the inclusion of comprehensive device information, and the capacity to be captured without necessitating a connection, make them particularly suitable for BLE device identification. Moreover, the scan response data provides additional data on devices. Therefore, BLE broadcast data and scan response data are mainly used in this work.

As mentioned before, it is crucial to understand the format of BLE data that can be captured with Android APIs. In Android APIs, scanning of BLE devices returns ScanResult from the ScanCallback function, which contains ScanRecord. The ScanRecord contains the raw bytes of scanning results. By experimenting on a physical Android phone, it is found that the ScanRecord is comprised of BLE advertising data and scan response data. The comparison of ScanRecord with advertising data and scan response data collected on PC is shown in Table 1 and Tabel 2, indicating that the ScanRecord is the concatenation of advertising data and scan response data. Thus, it makes sense to utilize both advertising data and scan response data of BLE scanning in this work.

Android phone
ScanRecord
02011808ff7d02010303ffff0816eefd0101000f01 020a00170948554157454920574154434820475420326 52d434142

Table 1: Scan Record data captured on Android phone

PC	
Advertising data	02011808ff7d02010303ffff0816eefd0101000f01
Scan response data	020a00170948554157454920574154434820475420326 52d434142

Table 2: Advertising data and scan response data captured on PC

There is no known available BLE link layer dataset that is suitable for BLE device classification. Therefore, a labeled BLE link layer dataset is part of the contribution of this work. To collect BLE link layer data on a PC, specific hardware and software are necessary to enable the successful capture of BLE packets. For this project, a USB dongle sniffer equipped with a nRF52832 SoC is selected, which is depicted in Figure 16. And Wireshark [1], a widely recognized tool, is utilized to facilitate data capture and analysis.

In this work, a diverse collection of 30 devices, spanning 13 different types, are served as data sources. Each device underwent a dedicated monitoring period to collect a sufficient volume of data frames. The specifics of the device capture list can be found in Appendix B. In an effort to maintain balance across the different device types, a nearly equivalent amount of data frames are collected for each category. Leveraging the combination of a USB sniffer and Wireshark, BLE advertising data and scan response data were successfully captured, generating pcapng files for subsequent analysis.

[1] https://www.wireshark.org/

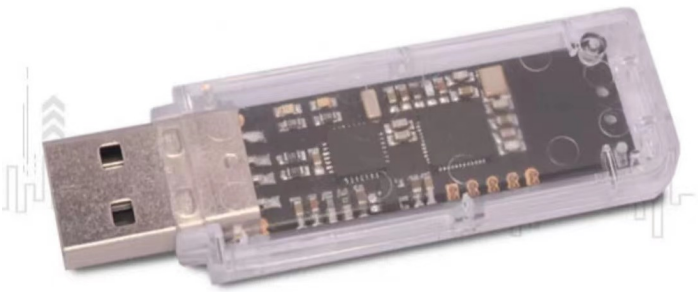


Figure 16: USB dongle with a nRF52832 SoC.

4.2 Data Analysis

In this section, the focus will be on the analysis of the gathered BLE link layer packets, exploring the information they encompass. The aim is to determine if this information is both adequate and valuable for the classification of BLE devices.

In section 4.2.1, different AD types will be analyzed to decide if sufficient information can be retrieved from link layer packets for classification purposes.

In section 4.2.2, the collected data from trackers will be analyzed, as they are main threats to privacy and security among BLE devices.

4.2.1 AD analysis

Upon careful examination of BLE advertising data and scan response data, it can be concluded that it can be effectively utilized for identifying BLE device types, as supported by the following key findings. Firstly, the frequent transmission of these two types of packets by devices makes them ideal for enabling the real-time detection and identification of nearby BLE devices. Secondly, the various types of Advertising Data (AD) contained within these packets possess distinct characteristics (outlined in Table 3). These unique traits provide sufficient information that can be leveraged for classifying and identifying BLE devices.

Flags AD analysis

The Flags AD field indicates the basic capabilities of the BLE device and gives other devices an understanding of how to interact with it. This field contains 8 bits of information, each indicating a specific capability. The first bit, when set, indicates that the device is in the LE Limited Discoverable Mode, typically applied for devices open to discovery for a

Table 3: BLE AD type.

AD type	**Description**
Flags	Flags field that indicates the interaction capabilities of device
Manufacturer Specific	Information or data specified by the manufacturer
Service Data	Data of services provided by the device
Service UUID	Unique identifier of services provided by the device
TX Power Level	Transmitted power level of the packet
Appearance	External appearance of the device
Local Name	Local name assigned to the device

Table 4: Description of each bit of Flags AD. Adapted from [4].

Data Type	**Octet**	**Bit**	**Description**
Flags	0	0	LE Limited Discoverable Mode
	0	1	LE General Discoverable Mode
	0	2	BR/EDR Not Supported.
	0	3	Simultaneous LE and BR/EDR to Same Device Capable (Controller)
	0	4	Simultaneous LE and BR/EDR to Same Device Capable (Host)
	0	5..7	Reserved for future use

brief time frame. The second bit, when flagged, indicates the device's status is in the LE General Discoverable Mode, meaning the device is openly advertising its presence to other devices in its proximity, ready to establish connections. The third bit denotes whether the device supports the Basic Rate/Enhanced Data Rate (BR/EDR) functionality, a feature of Classic Bluetooth mode. If this bit is set, it implies the device does not support BR/EDR, which is commonly the case with devices that exclusively support BLE. The fourth and fifth bits indicate simultaneous LE and BR/EDR to the same device capable of controller and host. The remaining bits from bit five to bit seven are reserved for future use. Table 3 summarizes the description of each bit. The most notable bit is the third bit. For those BLE devices that only support BLE and don't support Classic Bluetooth, this bit can be used as an important sign to distinguish it from other devices.

By interpreting the Flags AD, it becomes possible to gain insights into the basic capabilities of a BLE device and determine the strategy for establishing a connection.

Manufacturer Specific AD analysis

This data segment allows manufacturers to include specific information that may not conform to the standard BLE specifications, ensuring that their unique capabilities and features can be communicated to other devices within the network.

This data follows a two-octet company identifier, which is assigned by the Bluetooth Special Interest Group (SIG) [1]. This identifier ensures that devices receiving the information recognize and appropriately respond to the manufacturer-specific data.

This type of AD provides the manufacturer information contained in the identifier and the manufacturer-specific payload, which can be exploited in the identification process.

Service Data AD analysis

The Service Data AD type consists of a service UUID and the data associated with that service. The service UUID is typically 16-bit, 32-bit, or 128-bit, following which is the service data.

Information about services provided by the device can be extracted from this type of AD to be utilized in classification.

Service UUID AD analysis

The Service UUID AD includes a list of service or service class UUIDs. The key difference between these UUIDs and those in the Service Data AD is their use. Here, the UUIDs correspond to the services provided by the device, whereas in the Service Data AD, they typically represent the manufacturer's UUID.

This type of AD allows extraction of the types of services provided by devices, an important indication of the type of the device.

Local Name AD analysis

Some devices incorporate their local name within the advertising packets or scan response packets. If the device type is present in the local name, this AD type will be a strong indicator of the device type.

4.2.2 Analysis of collected data from trackers

Trackers like Airtags are crucial devices that should be analyzed considering the security issues. Thus, a thorough analysis of the link layer packets of some typical trackers will be undertaken in this section. The objective is to discover whether there are any unique attributes that can be extracted to aid their identification.

AirTag

Apple has implemented extensive measures to protect users' security and privacy. One notable action is the strategic concealment of information exposed in Airtags' advertising packets. This privacy-centric approach aligns with our findings from the collected BLE link layer packets, as demonstrated by the examination of representative packets from Airtags, presented in Appendix A.1. The advertising packet includes just one Advertisement Data (AD) - Manufacturer Specific. It indicates the manufacturer of this device is Apple and carries the data specified by the company.

Observations indicate that the advertising packets from AirTag exhibit distinct patterns depending on their state. The state of an AirTag can be categorized into three types: 'Unpaired', 'Nearby', and 'Lost'. The 'Unpaired' state refers to an AirTag that has not yet been established and bonded by a user. The 'Nearby' state indicates an AirTag in close proximity to its bonded user. Conversely, the 'Lost' state indicates an AirTag that is significantly distanced from its bonded user. In terms of safety implications, the 'Unpaired' state could be considered the least threatening, while the 'Lost' state has the highest potential for security concerns. A demonstration and comparison of the advertising packets of AirTag in different states is shown in Appendix A.5. The 'Nearby' state distinguishes itself from the other two states by the significantly shorter length of the packet. The difference between the 'Unpaired' state and 'Lost' state lies in the manufacturer specific data. Upon reverse-engineering the manufacturer specific data, it was found that the starting byte of the payload is a way to distinguish them. The starting byte of an AirTag in an unpaired state is typically 0x07, while the starting byte of the lost state is typically 0x12.

Nutale

Nutale, a tracker that works within Apple's Find My network, exhibits a unique pattern of link layer packets compared to AirTag. A thorough examination of Nutale's link layer packets is illustrated in Appendix A.2. Similar to AirTags, Nutale's packets maintain a short total length and feature an equal number of Advertisement Data entries. However, a distinction lies in the AD type, as Nutale incorporates Service Data within its advertising packets, diverging from the pattern observed in AirTags.

HuaweiTag

An analysis of the link layer packets of Huawei Tag reveals they are very similar to AirTag's packets. The interpreted result of the typical link layer packets of Huawei Tag by Wireshark and characteristics of Huawei Tag's packet are displayed in Appendix A.3. Attributes of link layer packets, such as total length, number of Advertisement Data, Advertisement Data type, and Advertisement length are the same as AirTag packets. However, the company ID and manufacturer-specific advertising data of Huawei Tag distinguish it from AirTag.

The patterns of three different states identified in the AirTag are similarly observed in the Huawei Tag, as shown in Appendix A.6. However, identifying between the three states of a Huawei Tag differs slightly from the methodology applied to the AirTag. Much like the AirTag, the 'Nearby' state of a Huawei Tag is distinguished from the other two states by a notably shorter packet length. However, the 'Unpaired' state distinguishes itself from the 'Lost' state through the presence of the initial two bytes as '0x0503', whereas the 'Lost' state typically begins with '0x2019'.

Tile

The link layer packets of the Tile tracker exhibit noticeable differences from those of the Airtag and HuaweiTag. A comprehensive breakdown of the link layer packets of Tile is presented in Appendix A.4. Notably, the amount of Advertisement Data Structure surpasses that of the other two devices. Despite this, the overall packet length remains consistent with the other two trackers. This is because all of them are relatively short due to the absence of scan response data.

4.2.3 Analysis of collected data from non-tracker devices

As described in Section 4.2.2, several typical trackers share common characteristics within their link layer packets. However, the scope of interest extends beyond these patterns to the analysis of non-tracker devices. This section presents an examination of select non-tracker devices, with a focus on exploring potential patterns for classification.

An initial attempt was made to identify patterns in non-trackers in general. One representative device from six distinct categories of non-tracker devices was selected for analysis, and the findings are displayed in Figure 17. The results reveal that the length of BLE link layer packets in non-trackers is generally greater than in trackers, and the number of Advertisement Data (AD) is also higher. This can be attributed to the inclusion of data in the scan response data (Scan_RSP) as indicated in Figure 17. In contrast, scan response data is frequently absent in trackers. Manufacturers of these devices often provide more information in their link layer packets than trackers, as non-tracker devices face fewer security concerns.

Driven by the successful identification of patterns among non-tracker devices in general, we will investigate the possibility of discovering patterns within device categories of finer granularity.

For this exploration, the 'Headphone' and 'Smart Watch' categories were chosen due to the greater availability of devices in these groups. Four representative devices from each category were selected for analysis and comparison. The analysis is depicted in Figure 18. No obvious pattern has been found in these two types to distinguish them utilizing packet characteristics. Another potential feature to be exploited in the BLE link layer packets could be Local Name AD. However, it is found that this AD is absent in some devices, making it unsuitable to be utilized to identify these two types.

4.3 Prototype Design

In this section, a feature based prototype is proposed, considering the specific information available in the BLE link layer packets. Several features from the BLE link layer packets are manually extracted, and the design of a SVM classifier, a Random Forest classifier and a Multi-layer Perceptron to exploit the manually selected features is illustrated.

Comparison of representative link layer packets of non-trackers						
Device Local Name	**Device type**	**Raw hex string of ADV_IND packet**	**Raw hex string of Scan_RSP packet**	**Total length (Bytes)**	**Number of AD**	**AD type list**
Huawei Watch GT2	Smart Watch	02011808ff7d020103 03ffff0816eefd010100f01	020a00170948554157 454920574154434820 47542032652d434142	48	5	[Flags, Manufacturer Specific, Service Data - 16 bit UUID, Tx Power Level, Device Name]
Samsung Galaxy Buds Live	Headphone	02011809ff75004016 fd6c648211073a048f 2ba97ea6ae1b49c619 e973a4a7	1b0947616c61787920 42756473204c697665 20284644364329204c 45	55	5	[Flags, Manufacturer Specific, 128-bit Service Class UUIDs, Device Name]
Haier Oximeter	Oximeter	0201061bff2909c81c 16102233d584fc4233 7754a09bc225cd90ad 48bca00a	0909552d424c444f585 9	37	3	[Flags, Manufacturer Specific, Device Name]
Xiaomi Blood Pressure Meter	Blood Pressure Meter	0201060f1695fe3058 b81f004c939bcc5cc8 08	1109696865616c7468 2e62706d2e62707831	37	3	[Flags, Manufacturer Specific, Service Data - 16 bit UUID, Tx Power Level, Device Name]
Yuwell Glucose Meter	Glucose Meter	020106070218d6081 80a180cff000000000 0d00343000152	0512c8004006020a00 0f09597577656c6c204 76c75636f7365	49	6	[Flags, Manufacturer Specific, 16-bit Service Class UUIDs, Tx Power Level, Device Name, Slave Connection Interval Range]
Huawei Scale 3	Scale	02010407ff72fddcb22 288	130948554157454920 5363616c6520332d30 3839020a04	34	4	[Flags, Manufacturer Specific, Tx Power Level, Device Name]

Figure 17: Analysis results of BLE link layer packets of non-tracker devices.

Considering the failure to find patterns in grouping devices into small categories as discussed in the last section, the classification granularity are decided to be (tracker, non-tracker).

Although the wealth of information embedded within the raw bit stream of BLE link layer packets, the high dimensionality of the raw advertising packets makes it difficult to be learned from the model. As the methodology evolves, the focus shifts towards a more refined approach to packet classification - feature extraction. The feature extraction approach selects specific, informative characteristics (or ”features”) from the BLE link layer packets as input for the classifier.

Several motivations drive the introduction of a model incorporating feature extraction. First, while raw packets contain abundant information, not all of this data is equally relevant for distinguishing device categories. Feature extraction concentrates on packet attributes most relevant to the classification task, potentially enhancing model performance. Furthermore, feature extraction can reduce data dimensionality, mitigating the ”curse of dimensionality”. This dimensionality reduction can result in simpler models that are more interpretable and less susceptible to overfitting.

To achieve this, meaningful features from the link layer packets are extracted and used as inputs to three different classification models - a SVM model, a Random Forest classifier and a Multi-Layer Perceptron (MLP) neural network. These models are selected due to their capability to manage complex, non-linear data and their versatility in model tuning,

Comparison of representative link layer packets of Headphone and Health devices

Device Local Name	Device type	Raw hex string of ADV_IND packet	Raw hex string of Scan_RSP packet	Total length (Bytes)	Number of AD	AD type list
Edifier Headphone	Headphone	02010607ff64687654566f030300790c094544494649455520424c45	/	28	4	[Flags, Manufacturer Specific, 16-bit Service Class UUIDs, Device Name]
Samsung Galaxy Buds Live		02011809ff75004016fd6c648211073a048f2ba97ea6ae1b49c619e973a4a7	1b0947616c6178792042756473204c69766520284644364329204c45	55	5	[Flags, Manufacturer Specific, 128-bit Service Class UUIDs, Device Name]
Xiaomi Buds Pro		02011a1bff8f03160114 8c42e4e45f00000005a46e46c76b7a010203040506	0dff2717080302505e6e6bc74610	45	3	[Flags, Manufacturer Specific, Manufacturer Specific]
Huawei Freebuds 4E		02010019 16eefd01010102c703000135040 10b470f01100011f3130641	/	29	2	[Flags, Service Data - 16 bit UUID]
Yuwell Glucose Meter	Health	020106070218d608180a180cff0000000000d00343000152	0512c8004006020a000f09597577656c6c20476c75636f7365	49	6	[Flags, Manufacturer Specific, 16-bit Service Class UUIDs, Tx Power Level, Device Name, Slave Connection Interval Range]
Xiaomi Blood Pressure Meter		0201060f1695fe3058b81f004c939bcc5cc808	11096968656 16c74682e62706d2e62707831	37	3	[Flags, Manufacturer Specific, Service Data - 16 bit UUID, Tx Power Level, Device Name]
Omron U728T		020106020a000302101808ff0e02012c000000	1e09424c45536d6172745f30303030303338444332323744324533335304632	50	4	[Flags, Manufacturer Specific, 16-bit Service Class UUIDs, Tx Power Level, Device Name]
Yuwell Glucose Meter		020106070218d608180a180cff0000000000d00343000152	0512c8004006020a000f09597577656c6c20476c75636f7365	49	6	[Flags, Manufacturer Specific, 16-bit Service Class UUIDs, Tx Power Level, Device Name, Slave Connection Interval Range]

Figure 18: Analysis results of BLE link layer packets of Headphone and Health devices.

making them appropriate for this classification task. The subsequent sections will focus on the feature extraction process and the details of each classification model.

Feature Engineering

Given the importance of identifying trackers in the task, features are first selected based on observation of trackers' packet characteristics. Four noteworthy patterns set these packets apart from others: packet length, the number of AD (Advertising Data) structures, a feature indicating support for Classic Bluetooth or not, and a feature indicating if 'Service Data' or Manufacturer Specific" AD structure is presented or not.

The length of the packet serves as the first potential feature. It's discovered that tracker packets are generally shorter in length compared to packets from other device categories,

as demonstrated in section 4.2.2. This difference in packet length suggests that length could be a promising feature for differentiating trackers from non-trackers.

Secondly, a lower count of AD structures within tracker packets is observed. The AD structure count provides crucial insights into the complexity and amount of information in each packet. Thus, a reduced number of AD structures may hint towards a more straightforward or less information-dense packet, which aligns with the characteristics of trackers as privacy and security are considered to be crucial in these devices.

Another observation is that tracker packets typically either lack support for traditional Bluetooth or suffer from missing information. This pattern of missing information or limited Bluetooth support can serve as an additional feature to discern tracker packets from those of other devices.

By observing the content of the Advertising Data structures within the packets of trackers, it's noticed that tracker packets typically contain either Manufacturer Specific Data or Service Data within their AD structures. This pattern provides an additional feature to consider, as its presence can serve as a strong indicator of tracker packets.

With the potential features identified, their distribution across both device categories: tracker and non-tracker is examined. This examination helps verify whether these features can offer a clear partition between the classes, crucial for successful classification.

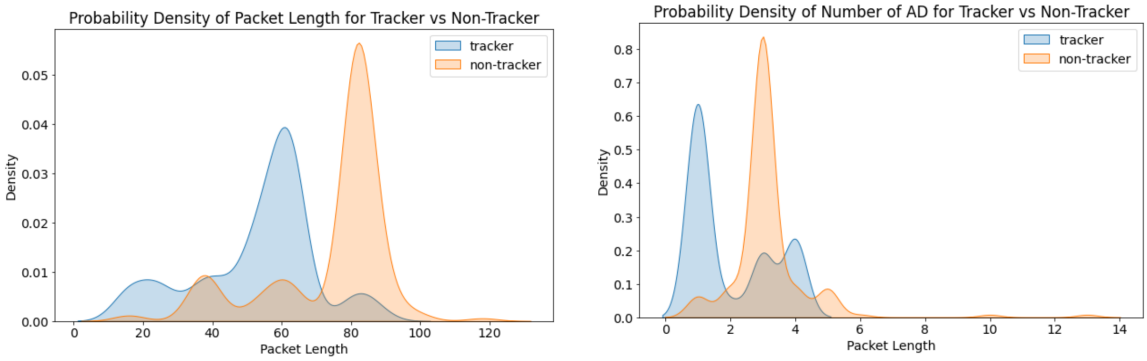
The probability density distributions of the packet length feature and the number of AD structure feature across the two device categories are visualized, as shown in Figure 19. Notably, the overlap between the two distributions is not significant and the peak of the two distributions lies in different regions in these two features. This suggests that these features could play a critical role in distinguishing between the classes.

As Classic Bluetooth feature and presence of manufacturer-specific or service data feature are categorical features, the feature distribution across tracker and non-tracker categories is visualized in the form of a bar chart in Figure 20. The figure shows that no trackers support Classic Bluetooth and a non-tracker can be either Classic Bluetooth supported, not supported, or unknown when there is a flag AD in the packet. It indicates the Classic Bluetooth feature can be useful in distinguishing trackers from non-trackers. Conversely, both categories of devices typically incorporated either Manufacturer Specific or Service Data in their packets. This makes it less likely to contribute effectively to the differentiation of device categories.

Given these observations, packet length, the number of AD (Advertising Data) structures, and Classic Bluetooth feature are selected as features for models.

4.3.1 Data Preprocessing

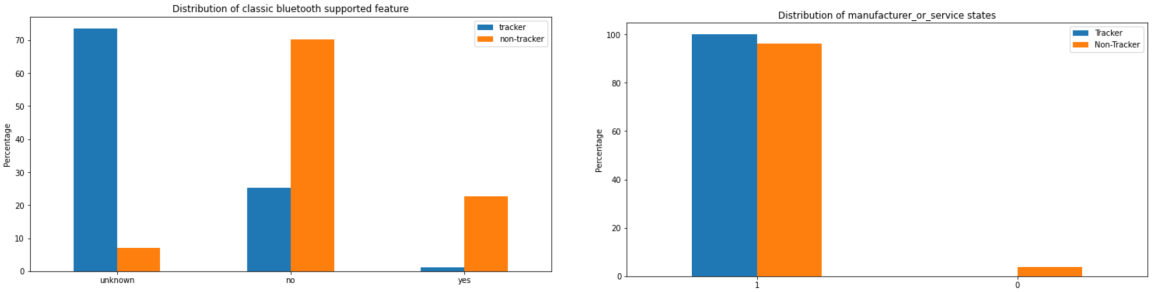
After feature engineering, the process continues to data preprocessing. BLE link layer packets were captured using Wireshark, producing pcapng files. These pcapng files were then parsed using Pyshark, a Python library proficient at pcapng file parsing. The outcome of this parsing was a dataset encompassing the raw bit stream of link layer packets,



(a) Box plot of distribution of length of packet across three categories.

(b) Box plot of distribution of number of AD across three categories.

Figure 19: Probability distributions of length of packet and number of AD feature.



(a) Box plot of distribution of Classic Bluetooth feature across three categories.

(b) Box plot of distribution of length of packet across three categories.

Figure 20: Box plot distribution of manufacturer or service feature and manufacturer or service feature.

three previously selected features and packet labels, designating whether a packet originates from trackers.

4.3.2 Feature-based Prototype

Support Vector Machine Classifier

The Support Vector Machine is a robust machine learning model, boasting several merits. It excels in remains resilient against noise, Moreover, SVM can handle linear and non-linear data while being an innate binary classifier. Therefore, an SVM model was chosen to classify devices into the trackers, non-trackers category.

Prior to training the SVM model, standardization was applied to the data samples to ensure zero mean and unit variance. This is pivotal as SVM, being a distance-centric algorithm, is sensitive to feature scale. Absent standardization, dominant features could overpower the model, leading to sub-par performance. Moreover, standardizing data improves the model's numerical stability and expedites the optimization algorithm's convergence inherent in SVM.

Hyperparameter	Value list
C	[0.01, 0.1, 1, 10]
degree	[2, 3, 4, 5]
coef0	[0.0, 0.1, 0.5, 1.0]
gamma	['scale', 'auto', 0.1, 1, 10, 100]

Table 5: Hyperparameters list for grid search in SVM.

For this specific task, the Polynomial kernel function was selected. This choice was inspired by the potential complex and polynomial relationship between dataset features and the target variable. The Polynomial kernel allows non-linear classification, projecting data into higher dimensions where classes can be more easily distinguished. The function possesses two vital parameters: the polynomial's degree and the coefficient 'C'. The degree dictates model complexity, while 'C' steers the balance between permitting training errors and demanding stringent margins.

Considering the dataset's limited size, creating a separate validation set was deemed unnecessary. A validation set might inadequately reflect data distribution, culminating in sub-optimal model calibration and potentially misleading evaluation metrics. Instead, a more comprehensive strategy was implemented through grid search.

Grid search is a method of hyperparameter tuning that tests and evaluates a model for every possible combination of hyperparameters within a predefined list. Notably, during grid search, each iteration of model training incorporates cross-validation rather than solely relying on a singular hold-out validation set.

In cross-validation, training data gets segmented into 'k' subsets or 'folds'. The model then undergoes training 'k' times, each iteration using 'k-1' folds for training and the left-out fold for validation. The average of the values computed in this loop becomes the performance metric presented by the k-fold cross-validation. This technique ensures maximal utilization of the limited data available for both training and model validation, giving a far more accurate estimate of model performance compared to a single validation set. This method is especially valuable when handling datasets of limited size.

Using grid search and cross-validation, an exploration of values was conducted for the SVM classifier's hyperparameters with the polynomial kernel. This exploration covered parameters such as 'C', 'degree' for the polynomial kernel function ('poly'), 'coef0' - an independent term in the kernel function, and the 'gamma' parameter. This exhaustive, automated search identified the optimal parameters, resulting in the highest cross-validated performance on the training data and thus improving the final model's ability to generalize to new, unseen data. The hyperparameters used for grid search can be found in Table 5.

Random Forest Classifier

The selection of a machine learning model for the classification of BLE link layer packets based on the extracted features leans towards a Random Forest classifier due to several reasons.

Firstly, given the absence of a known public dataset and the difficulty of collecting enough BLE devices to provide a large dataset, overfitting poses a significant challenge for this classification task. The Random Forest classifier, as an ensemble method, provides a robust means of dealing with overfitting. By creating multiple decision trees and combining their output, the model can maintain high accuracy and generalizability, even with complex feature sets. Secondly, Random Forest, with the randomness introduced during the training process, is robust to outliers and noise in the dataset, preventing them from significantly impacting the performance of the model.

While employing standardization as a pre-processing step and grid search for hyperparameter optimization, attention is dedicated to tuning key hyperparameters of the Random Forest Classifier. The following parameters are considered:

'n_estimators' denotes the number of trees in the forest. Each tree in the forest is trained independently, and the final prediction of the forest is an aggregation of the predictions from all trees. Having a large number of trees can reduce overfitting, making the training process slower and requiring more memory.

'max_features' is the number of features to consider when looking for the best split. This can be a critical parameter to tune, as considering all features at each split can lead to overfitting. Two options, 'auto' and 'sqrt', are under our consideration.

'max_depth' controls the maximum depth of each tree in the forest. A tree's depth is the length of the longest path from the root to a leaf. A larger depth allows the tree to model complex patterns by creating more splits, but it also makes the model more prone to overfitting.

'min_samples_split' defines the minimum number of samples required to split an internal node, while 'min_samples_leaf' is the minimum number of samples required to be at a leaf node. These parameters help prevent overfitting by controlling how deep our tree can grow.

'bootstrap' parameter indicates whether bootstrap samples are used when building trees. Bootstrap sampling can help reduce variance and overfitting.

'criterion' is the function to measure the quality of a split. We have considered 'gini' for Gini impurity and 'entropy' for the Shannon information gain.

Through an exhaustive grid search and optimization process, the optimal combination of these parameters delivering the best performance for the Random Forest Classifier can be identified. A table of these parameters is presented in Table 6.

Hyperparameter	Value list
n_estimators	[10, 50, 100, 200]
max_features	['auto', 'sqrt']
max_depth	[10, 20, 30, None]
min_samples_split	[2, 5, 10]
min_samples_leaf	[1, 2, 4]
bootstrap	[True, False]
criterion	['gini', 'entropy']

Table 6: Hyperparameters list for grid search in Random Forest Classifier.

Multi-Layer Perceptron Classfier

Multi-Layer Perceptron (MLP), a type of artificial neural network, serves as the second classifier employed in this prototype. MLP, with its multiple layers and non-linear activation functions, exhibits high capability in modeling complex and non-linear relationships, which is often seen in practical classification problems. Again, we select the same features as explained in Random Forest classifier.

The three extracted features from BLE link layer packet: packet length of packet, number of AD structures, and Classic Bluetooth feature, may have intricate relationships in defining the device classes. The powerful non-linear modeling ability of MLP can thus be leveraged to capture these relationships, thereby distinguishing between different device types more effectively.

Moreover, MLP has the added advantage of efficiently handling high-dimensional data. Even though our current dataset consists of only three features, MLP's potential to deal with more features can facilitate the inclusion of additional characteristics in the future, enhancing the flexibility and scalability of our model.

Despite its strengths, MLP's performance heavily relies on the optimal configuration of several hyperparameters, including the number of hidden layers, the number of neurons in the hidden layers, and the type of activation function used. Consequently, we apply a similar hyperparameter optimization strategy as with the Random Forest classifier, using grid search to fine-tune these parameters for optimal model performance. We've considered several key parameters in our Multi-layer Perceptron (MLP) model for fine-tuning, as explained below.

The 'hidden_layer_sizes' parameter specifies the structure of the hidden layers in terms of layer count and neurons per layer. This impacts the model's capacity to learn complex patterns in the data. The various options allow the model to explore different balances between capacity and overfitting risk.

The 'activation' parameter sets the activation function used by the neurons in the hidden layers. This function introduces non-linearity into the model, allowing it to learn more than just linear relationships in the data. We considered 'tanh' and 'relu' because they are commonly used and generally perform well.

The 'solver' parameter specifies the optimization algorithm used to adjust the weights during training. We considered 'sgd' (Stochastic Gradient Descent) and 'adam' as they are two of the most popular optimization algorithms. The former is known for its robustness and the latter for its efficiency.

The 'alpha' parameter sets the rate of L2 regularization, which can help prevent overfitting by adding a penalty term to the loss function based on the magnitude of the weights. We've considered multiple values to find the right balance between underfitting and overfitting.

The 'learning_rate' determines how the weights in the network are updated. A 'constant' learning rate means that the learning rate does not change during training. 'adaptive' learning rate, on the other hand, means that the learning rate will decrease as the model gets closer to a solution, potentially leading to better final performance.

'learning_rate_init' is the initial learning rate when 'solver' is set to 'sgd' or 'adam'. Different learning rates can lead to significantly different training dynamics and results, so we've considered several common values to find the one that leads to the best performance.

A set of values for each hyperparameter was selected, as shown in Table 7. The optimal parameters for the model, determined after the grid search, will be discussed in Chapter 5.

Furthermore, to enhance model performance and generalization, batch normalization and dropout were integrated into the MLP structure. Batch normalization helps in achieving consistent activations throughout the training, thus accelerating convergence. Meanwhile, dropout serves as a regularization method, where a fraction of neurons are randomly dropped or deactivated during training, reducing the risk of overfitting.

Following the grid search, the optimal architecture for the hidden layers has been determined. The hidden layer comprises three layers, each with 10, 20, and 10 neurons respectively. Each hidden layer is followed by a batch normalization layer and a dropout layer to further enhance the model's resilience to overfitting. This structure forms the final architecture of the model, as depicted in Fig 21.

4.4 Integration of Classification Algorithm into HomeScout

Incorporating trained machine learning models into Android applications requires a systematic approach to ensure easy integration and real-time performance. TensorFlow Lite [2] is a mobile library for deploying models on mobile. It offers a lightweight solution that can run TensorFlow models on devices with limited computational resources. Furthermore,

[2]https://www.tensorflow.org/lite

TensorFlow Lite models are optimized for speed, ensuring quick inference times, which is essential for real-time applications on mobile devices. More importantly, TensorFlow provides an end-to-end workflow - from training the model using TensorFlow to deploying it on mobile devices using TensorFlow Lite.

In light of these advantages, classification model is rebuilt using TensorFlow, subsequently converting it to TensorFlow Lite format, which can be loaded into Android framework for seamless integration into HomeScout. The overall transition process is illustrated in Figure 22.

Data preprocessing and feature extraction were also replicated in implementation in HomeScout. After integration of proposed algorithm, HomeScout is able to classify between tracker and non-tracker. A demonstration of the implemented classification service in HomeScout is shown in Figure 23.

Hyperparameter	Value list
hidden_layer_sizes	[(10),(20),(50),(10,10),(10,20),(10,50),(10,10,10), (10,20,10),(10,50,10),(20,20,20),(20,40,20),(50,50,50)]
activation	['tanh', 'relu']
solver	['sgd', 'adam']
alpha	[0.0001, 0.001, 0.01]
learning_rate	['constant','adaptive']
learning_rate_init	[0.1, 0.01, 0.001, 0.0001]

Table 7: Hyperparameters list for grid search in MLP.

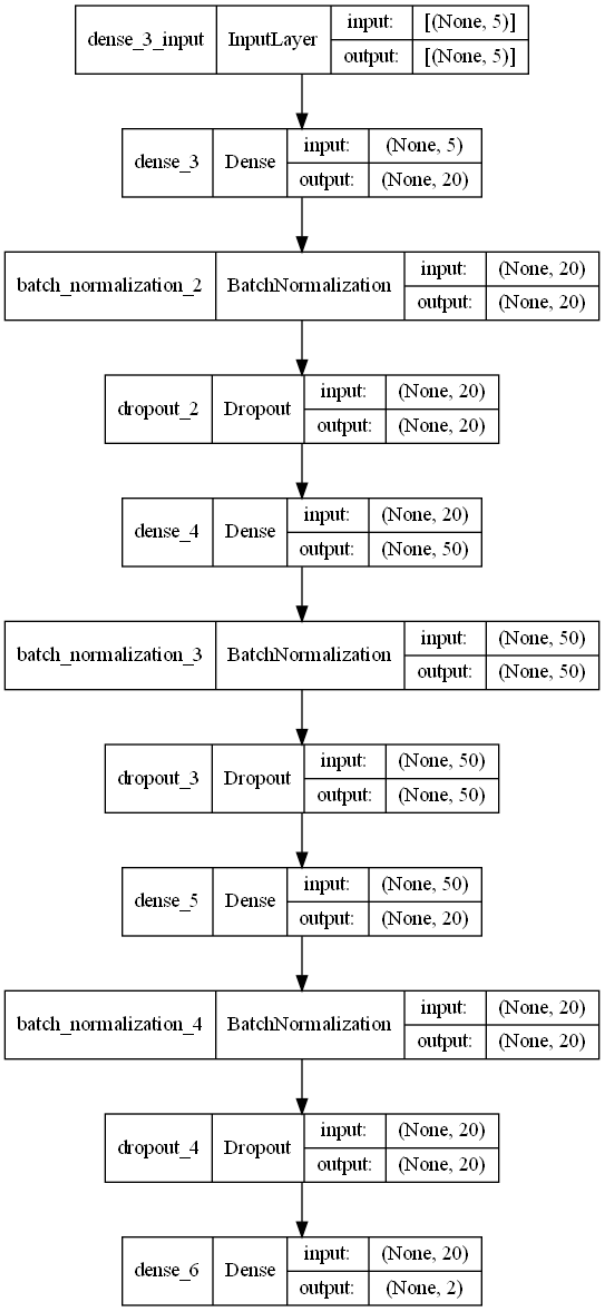


Figure 21: Network architecture of MLP model.

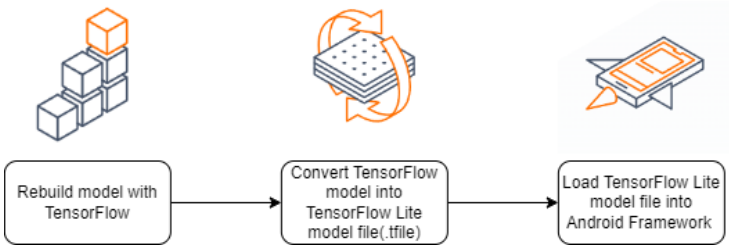


Figure 22: Integration process of model into HomeScout.

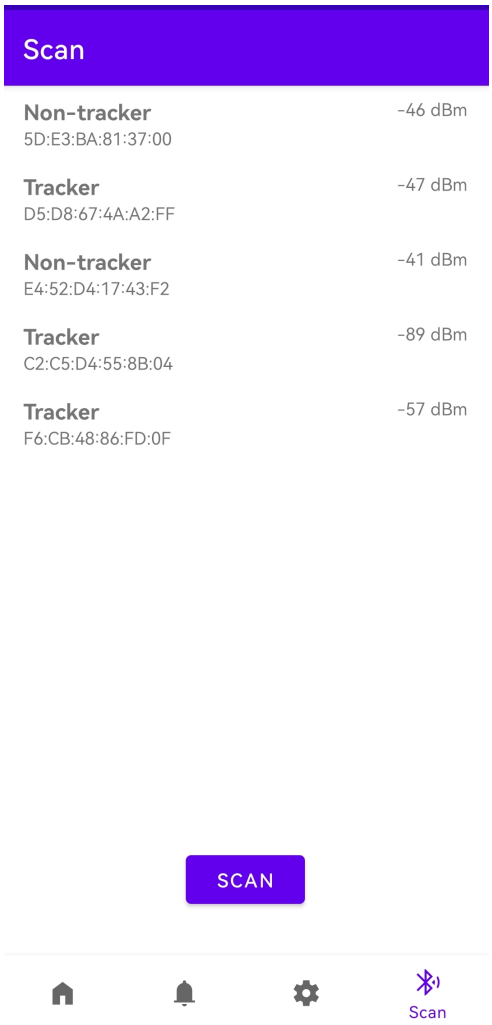


Figure 23: Integrated classification service in HomeScout.

Chapter 5

Results and Evaluation

In this chapter, the results and evaluation of the proposed feature-based prototype. Its capability in classifying BLE packets into pre-defined categories is demonstrated and discussed.

5.1 Evaluation Metrics Selection

Before evaluating the models, several metrics are selected to determine the capability of classification as follows:

- Accuracy: Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations.

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{Total Observations}} \tag{5.1}$$

- Precision: Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. It's also known as the Positive Predictive Value.

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \tag{5.2}$$

- Recall (Sensitivity): Also termed as Sensitivity or True Positive Rate, it represents the ratio of correctly predicted positive observations to all observations in the actual class.

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} \tag{5.3}$$

- F1-Score: The F1 Score is the weighted average of Precision and Recall. This score takes both false positives and false negatives into account and is particularly beneficial when the class distribution is uneven.

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.4}$$

Category	Number of Frames	Ratio
Tracker	89511	51%
Non-tracker	87639	49%

Table 8: Distribution of data frame in tracker and non-tracker category.

- Confusion Matrix: A table that describes the performance of a classification model. It illustrates the different types of correct and incorrect predictions made by a classifier. The confusion matrix offers a comprehensive picture of the classifier's performance, shedding light on not only the errors made by the classifier but also the nature of these errors.

5.2 Evaluation Dataset

Devices are categorized into two groups in this work: trackers and non-trackers. To ensure a balanced distribution of data samples across the two classes, the amount of data frame in two categories is counted and displayed in Table 8. The distribution is balanced, with approximating a 1:1 ratio between the two categories.

5.3 Evaluation of Feature-based Prototype

This section describes the performance evaluation of proposed three feature-based models in BLE packets classification, SVM classifier, Random Forest classifier and Multi-Layer Perceptron classifier.

5.3.1 SVM Classifier

5.3.2 Hyperparameters Fine-tuning

To find the best hyperparameters for the SVM model with a polynomial kernel, the grid search strategy was utilized. This technique involves training and validating the model on all possible combinations of the hyperparameters in a predefined list. By systematically exploring this parameter space, the combination that maximizes the model's cross-validated performance was identified.

The grid search was conducted over the following parameters:

- 'C': The regularization parameter, with smaller values specifying stronger regularization. This controls the trade-off between achieving a low training error and a low testing error, which is the ability to generalize the classifier to unseen data.

Category	Precision	Recall	F1-score
Non-tracker	0.99	0.93	0.96
Tracker	0.88	0.98	0.93

Table 9: Classification report of Support Vector Machine classifier

- 'degree': The degree of the polynomial kernel function. This parameter represents the power to which the sum of the product of the input vectors, along with the independent term 'coef0', is raised.
- 'gamma': The kernel coefficient, which can affect the shape of the decision boundary.

Following the grid search procedure, the optimal hyperparameters were identified as: C = 10, degree = 4, coef0 = 0.1, gamma = 'scale'. This indicates that a 4-degree polynomial kernel with a regularization parameter of 10, an independent term of 1.0, and a kernel coefficient 0.1 in the kernel function yielded the highest cross-validated performance.

5.3.3 Model Evaluation

In this section, the performance of the proposed SVM model with a polynomial kernel is evaluated.

Class 0, representing non-trackers, achieves a precision of 0.99 and a recall of 0.93, leading to an F1-score of 0.96. Class 1, which stands for trackers, exhibits a precision of 0.88 and with a recall of 0.98. The harmonic mean of precision and recall, or the F1-score, is thus 0.93. This demonstrates that the model has good performance in distinguishing between trackers and non-tracker in the dataset. A summary of these metrics are displayed in Figure 9

The macro average of precision, recall, and F1-score are around 0.93, 0.95, and 0.94, respectively, while the weighted averages are 0.95, 0.95, and 0.95, respectively, suggesting low overall performance.

The good performance is further supported by the confusion matrix as displayed in Figure 24. While only 7% of non-tracker instances have been misclassified in the test set, less (2%) of tracker instances have been misclassified as non-tracker.

5.3.4 Random Forest Classifier

Hyperparameters Fine-Tuning

Following the grid search over a predefined set of hyperparameters, an optimal set that enhanced the performance of the Random Forest Classifier was identified. The chosen hyperparameters are as follows:

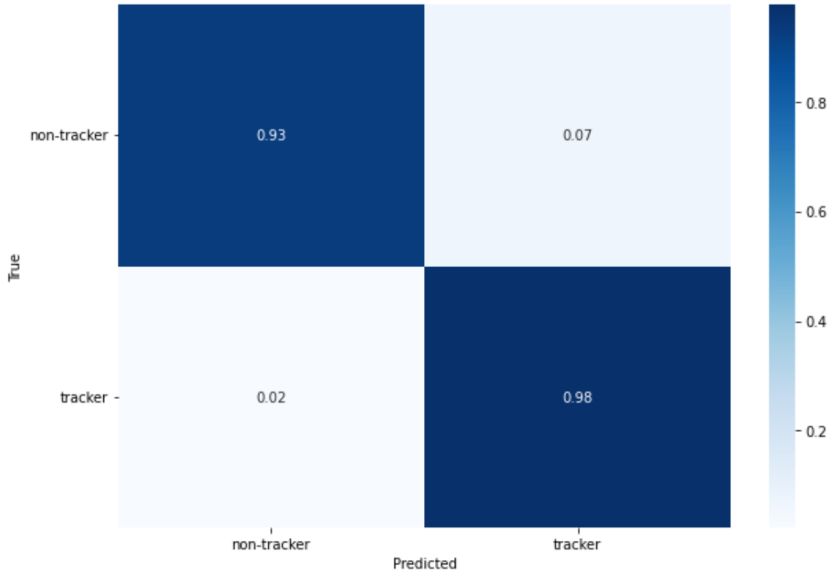


Figure 24: Confusion matrix of Support Vector Machine classifier.

- 'n_estimators': 10 - Increasing the number of trees can improve model accuracy up to a certain point, but can also lead to slower training times and overfitting. A value of 10 is selected to balance model accuracy and computational efficiency.

- 'max_depth': 10 - More depth can allow the model to learn more complex patterns by creating additional splits, but may also lead to overfitting. A value of 10 was chosen to control the complexity of the learned models and prevent overfitting.

- 'max_features': 'auto' - This determines the number of features to consider when looking for the best split. Setting this to 'auto' lets the model decide the optimal number of features to consider.

- 'min_samples_leaf': 2 - This parameter avoids creating trees where any leaf would have only a few samples, again controlling the complexity of the learned models and preventing overfitting.

- 'min_samples_split': 10 - A larger value ensures that the model is not overfitting the training data.

- 'criterion': 'Entropy' is chosen as the criterion to measure the impurity of an input set. This is consistent with our guess as Entropy tends to produce slightly more balanced trees than Gini Impurity. This is because it is more sensitive to differences in class probabilities.

- 'bootstrap': False - Whether bootstrap samples are used when building trees. When set to False, the whole dataset is used to build each tree.

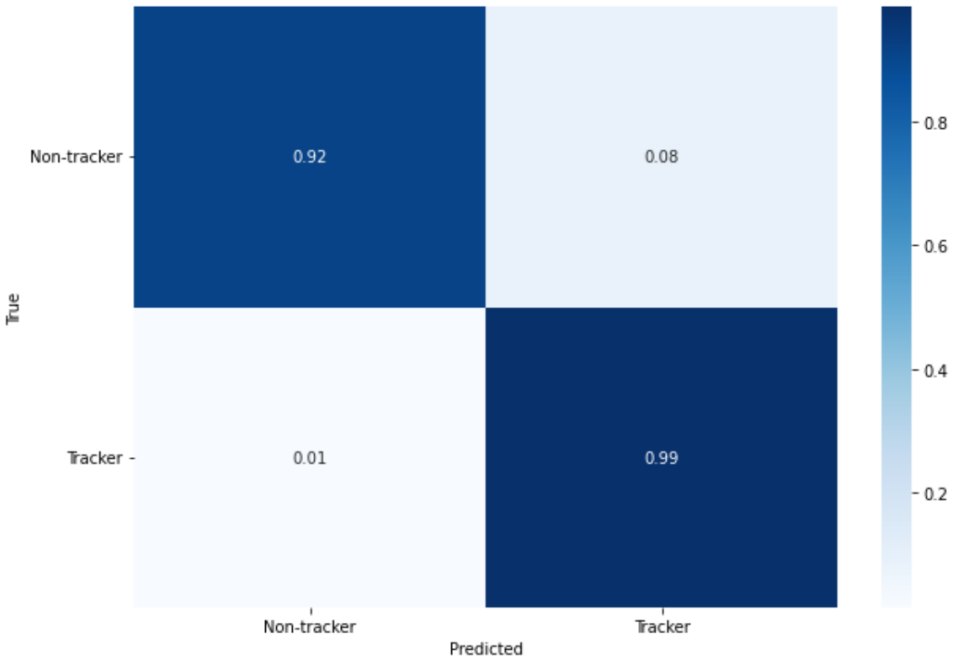


Figure 25: Confusion matrix of Random Forest classifier in feature-based prototype.

Model Evaluation

The model achieves an accuracy of 93.7% on test dataset, surpassing the proposed SVM classifier. Class 0, representing non-tracker devices, achieves nearly perfect precision (0.99) and high recall (0.92), leading to an F1-score of 0.95. Class 1, which stands for trackers, exhibits good precision (0.84) and a high recall of 0.99. The harmonic mean of precision and recall, the F1-score, is thus 0.90.

The weighted averages of precision, recall, and F1-score are all around 0.94, demonstrating impressive overall performance. These metrics are reported in Table 10.

The confusion matrix further supports these findings and is presented in 25. 92% of non-tracker is correctly classified. For tracker, only 1% is mistakenly predicted as non-tracker.

These results indicate that the Random Forest Classifier is highly proficient at distinguishing between non-tracker and tracker devices and have better classification accuracy on test set.

Category	Precision	Recall	F1-score
Non-tracker	0.99	0.92	0.95
Tracker	0.84	0.99	0.90

Table 10: Classification report of Random Forest classifier

5.3.5 Multi-Layer Perceptron Classifier

Hyperparameters Fine-Tuning

The grid search strategy identifies an optimal set of hyperparameters for the Multi-layer Perceptron model.

The 'activation' function selected was 'relu' (Rectified Linear Unit). The rectifier function is a popular choice as it introduces non-linearity without requiring expensive computations. It aids in mitigating the vanishing gradient problem, which is a common issue during backpropagation in deep neural networks, thereby improving the training phase's efficiency and effectiveness.

The model uses an 'alpha' value of 0.0001. The choice of alpha affects the compromise between underfitting and overfitting. An alpha value of 0.001 indicates that the model benefits from some regularization to prevent overfitting but is not so strong as to oversimplify the model to the point of underfitting.

The 'hidden_layer_sizes' determined to be optimal were (20, 40, 20). This indicates a network structure with three hidden layers, each comprising 10 neurons. This configuration allows the model to learn a sufficient level of complexity from the data without overcomplicating the network and risking overfitting.

The 'learning_rate' parameter was set to 'constant'. A constant learning rate implies that the step size in weight remains the same throughout the training process. This constant learning rate suggests that an adaptive learning rate may not be necessary for this dataset and task.

The 'learning_rate_init' was set at 0.1, a relatively high value indicating that the model takes relatively large steps during the initial iterations of weight updates in backpropagation. It implies that the model converges faster and doesn't require a very slow initial learning process.

Lastly, the 'solver' parameter was set to 'sgd' (Stochastic Gradient Descent). SGD is a widely used optimization algorithm known for its robustness, especially when dealing with large and complex datasets. It suggests that for this task, SGD provided better or similar results compared to other optimization algorithms like 'adam'.

This tuned MLP model leverages these optimal hyperparameters to train and predict effectively, resulting in enhanced model performance.

Model Evaluation

The evaluation metrics of MLPs on the performance of test set are examined in this section.

The model achieves an accuracy of 94.5%, surpassing the proposed Random Forest Classifier. It exhibits a nearly perfect precision (0.99) for non-tracker, and high precision

Category	Precision	Recall	F1-score
Non-tracker	0.99	0.93	0.96
Trackers	0.88	0.98	0.93

Table 11: Classification report of Multi-Layer Perceptron classifier.

(0.88) for tracker, indicating reliable predictions for these classes. Recall rates further demonstrate this model's capabilities, with non-tracker at 0.93 and an even higher rate of 0.98 for tracker, indicating the model's proficiency in correctly identifying class instances. The F1-scores, 0.96 for non-tracker and 0.95 for tracker, further prove the model's accuracy and reliability by balancing the precision and recall. A summary of these evaluation metrics is provided in Table 11.

In conclusion, the Multi-layer Perceptron classifier generally outperforms the Random Forest classifier in classifying the selected features on test set. The superior performance of the Multi-layer Perceptron (MLP) over the Random Forest Classifier can be attributed to several reasons tied to the nature of MLPs and the characteristics of the dataset.

MLPs are more suitable for datasets with non-linear relationships due to their architecture, which allows them to learn and model non-linear interactions between variables. Conversely, random forests may struggle with complex non-linear relationships.

In an MLP, all features interact with each other through weights in the hidden layers, allowing the model to capture complex patterns within the data. In contrast, Random Forests consider features independently, which could potentially limit their ability to detect intricate patterns that rely on multiple feature interactions.

5.4 Experiments on Unseen Device

In prior evaluations, the emphasis was on the model's proficiency in classifying data within the same group of devices, namely intra-device classification capability. This approach, while essential, does not paint the full picture of a model's capabilities. Real-world scenarios often demand the identification of unfamiliar devices, and the model must be equipped to handle this challenge. Consequently, the next phase of experimentation will focus on 'inter-device classification', where the model will be evaluated based on its capacity to classify data from devices it hasn't encountered before. This phase is crucial to ascertain the model's generalization capabilities, revealing its performance on dynamic and diverse world of devices.

To conduct such experiments, 10 devices are collected to serve as the evaluation dataset. Each device underwent a dedicated monitoring period to collect a sufficient volume of data frames. The specifics of the device capture list can be found in Appendix B.

The three proposed models will be further assessed to determine their generalization strength, ensuring its suitability for deployment in HomeScout.

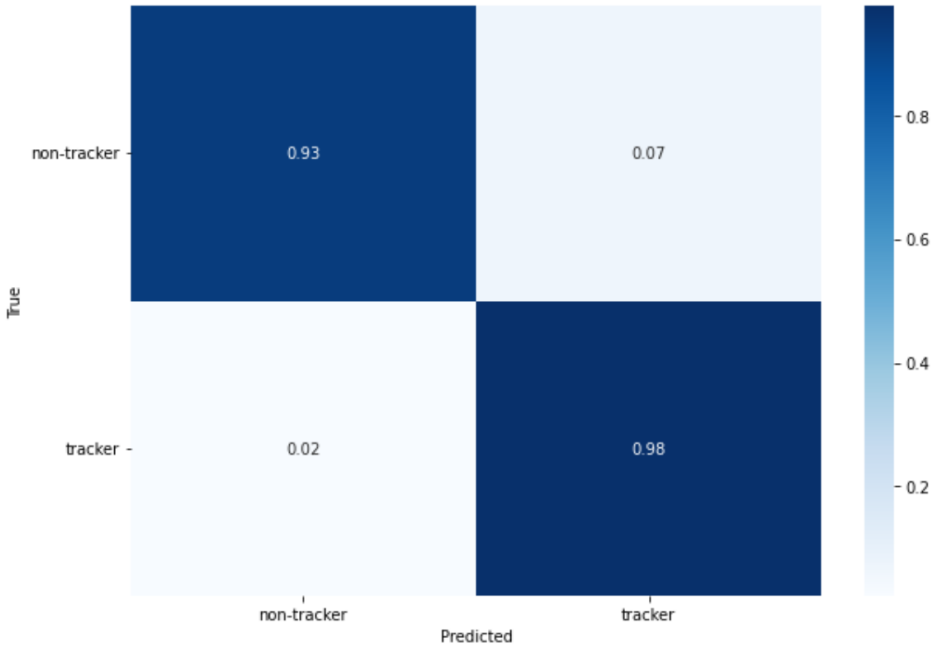


Figure 26: Confusion matrix of Multi-Layer Perceptron classifier in feature-based prototype.

Category	Precision	Recall	F1-score
Non-tracker	0.99	0.92	0.95
Tracker	0.87	0.98	0.92

Table 12: Classification report of Multi-Layer Perceptron classifier on unseen data.

The Multi-layer Perceptron (MLP) model demonstrated superior performance on the unseen device test set, achieving an accuracy of 88%. In comparison, the SVM model attained 68% accuracy, while the Random Forest Classifier reached 80%. This underscores the effectiveness of the MLP model in this context. The classification report and confusion matrix of the Multi-layer Perceptron for this test are presented in Figures 12 and 27, respectively. The model posted an accuracy rate of 88%, which aligns close to the results from the intra-device classification on the test dataset. Based on these results, the Multi-layer Perceptron model is proved to be more reliable and suitable to be deployed in HomeScout.

5.5 Validation Experiments on Homescout

To further validate the enhanced classification capabilities of HomeScout, specifically its ability to distinguish between trackers and non-trackers after integrating the proposed model—an experiment was conducted. In this test, 14 devices were evaluated using HomeScout, of which only 2 were inaccurately predicted. This translates to an accuracy of 85.7%, consistent with the results presented in the previous sections. Detailed

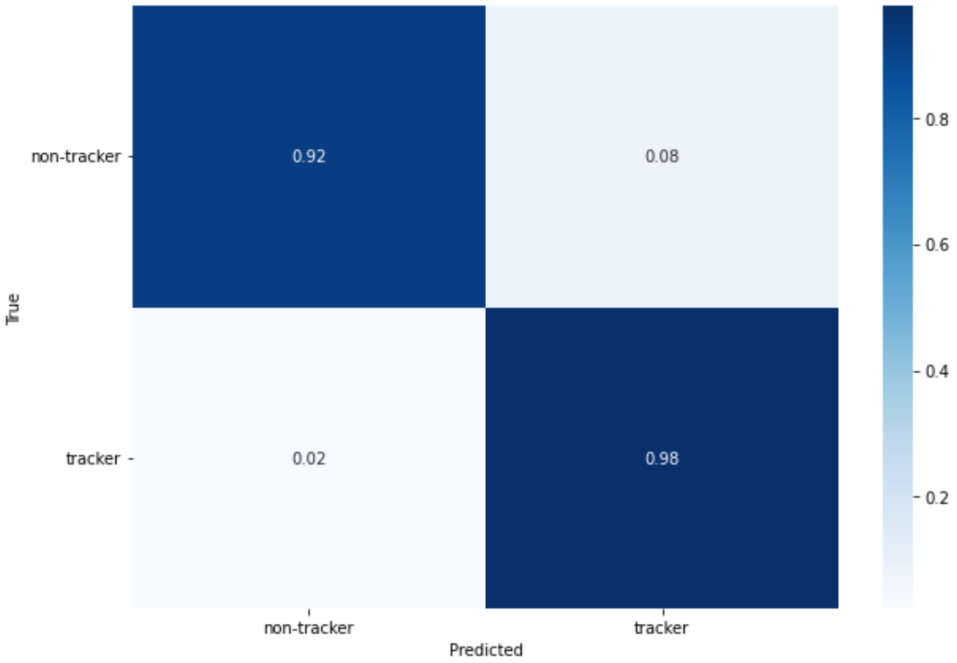


Figure 27: Confusion matrix of Multi-Layer Perceptron classifier on unseen data.

classification outcomes can be found in Table 13. This validates that the integration of the proposed model has indeed enhanced HomeScout's identification capability. It underscores the model's adaptability and generalization capabilities when applied in real-world scenarios, rather than just controlled environments.

Device name	MAC address	Predicted Label	Ground Truth
Tile Slim	C0:B9:F6:A7:E5:6E	Tracker	Tracker
Mangotek	F6:CB:48:86:FD:0F	Tracker	Tracker
AirTag	EE:C1:4C:D9:5C:12	Tracker	Tracker
HuaweiTag	C1:67:DA:10:04:12	Tracker	Tracker
Nut Focus	C5:2E:E4:02:69:8D	Non-tracker	Tracker
Huawei Watch GT2	34:B2:0A:95:8C:AB	Non-tracker	Non-tracker
Keep S2	34:B2:0A:95:8C:AB	Tracker	Non-tracker
Omron Blood Glucose Device	F0:5E:CD:4A:59:3E	Non-tracker	Non-tracker
Huawei Scale 3 Pro-037	88:22:B2:CB:F6:F6	Non-tracker	Non-tracker
Huawei Watch 8	40:DC:A5:18:DE:FA	Non-tracker	Non-tracker
Logitech Pebble Mouse	E7:2C:D0:C9:9A:D3	Non-tracker	Non-tracker
Lepu Oximeter	E9:D1:6B:E5:2D:46	Non-tracker	Non-tracker
Yuwell Blood Pressure Meter	44:28:A3:62:E7:68	Non-tracker	Non-tracker
Dell Mouse	EA:54:3A:7D:E9:7A	Non-tracker	Non-tracker

Table 13: Classification results of experiment on HomeScout

Chapter 6

Conclusion and Future Work

This chapter serves as a conclusion to this work, summarizing the main findings and contributions. Additionally, this chapter also provides insights for researchers who wish to pursue further research in this field.

6.1 Conclusion

In this work, we have addressed the pressing security and privacy concerns associated with Bluetooth Low Energy (BLE) devices, particularly location trackers, by developing a comprehensive method for classifying BLE devices. The motivation behind our research was driven by the need to bridge the gap in security measures between iOS and Android users and to empower Android users with a powerful tool to detect and categorize various BLE devices encountered in their daily lives.

Research objectives were successfully achieved through a systematic and methodical approach. This work thoroughly examined the BLE link layer communication and collected a diverse dataset comprising 30 BLE devices. Several unique patterns were identified from these collected link layer packets and then used as input features for machine learning models. A feature based prototype was proposed. Four features were evaluated and three featueres were selected to be the final feature list. Three machine learning models were implemented to pick the best performing one. The Multi-Layer Perceptron model demonstrated a superior performance over the SVM model and the Random Forest model, thus becoming our final model of choice. This model effectively distinguishes between tracker and non-tracker with a 94.5% of accuracy in the test set and 88% on unseen device test set, providing users with increased awareness and control over their surrounding BLE devices.

Before the integration of the classification algorithm into HomeScout, a bug was identified in the application. The AppIntro library was found to be unreliable for handling permission requests. On certain Android devices, the application encounters difficulties in successfully obtaining permission and repeatedly redirects to the app intro page in

a recursive manner. It was finally replaced by XXPermissions library and successfully address this problem.

Following this fix, the classification algorithm was integrated into the HomeScout application. The integration with TensorFlow Lite has significantly extended its capabilities, enabling the detection and classification of personal tracker and non-tracker. This enhancement has provided Android users with an equivalent level of protection to their iOS counterparts against potential tracking threats.

6.2 Future Work

The classifier built for this work is a prototype, hence it is not ready for the market. It can be used to classify BLE devices into three categories, but further improvements are necessary.

While our research objectives were achieved through a systematic approach, we recognize that one of the significant limitations of this work is the scale of the dataset. The dataset used for training and evaluation was relatively small, comprising a limited set of BLE devices from various types. To ensure the generalization capability and robustness of the classification model, it is essential to include a more extensive and diverse dataset consisting of a wide range of BLE devices.

To address this limitation, further research efforts to expand the dataset are encouraged. Incorporation of a more diverse array of BLE devices from different manufacturers, regions, and applications can be essential. This expansion will not only allow the algorithm to classify into more categories but will also provide a refined granularity of those categories. Such detailed classification is crucial for practical applications, as it ensures that the system can accurately identify and distinguish between various BLE devices. Furthermore, the creation of a public dataset can be beneficial for the research community. However, careful consideration must be given to ethical concerns and data anonymization to ensure that user privacy and sensitive information are adequately protected.

Another constraint in this work is the challenge presented by MAC randomization. Within the current framework, MAC randomization hasn't been considered. If a BLE device changes its MAC address post-detection by HomeScout's scanning service, it will be recognized as a separate entry, causing the application to interpret it as two different devices. To address this issue, a mechanism that can track devices undergoing MAC randomization needs to be developed, ensuring that they are consistently identified as the same device regardless of address changes, thus preventing redundant entries in the device detection list of the application.

To further improve the performance of the classification, it's worthwhile to consider incorporating an additional prospective feature in the feature engineering process, which is the inter-arrival-time distribution of BLE advertising packets. This feature has been demonstrated effective in classifying Bluetooth Classic packets of Wearable in [6]. The inter-arrival-time often captures the unique temporal patterns of devices. Different devices can have distinct patterns of sending data, making this metric valuable for identification.

However, the classification algorithm was initially developed and trained on a Windows platform before being integrated into the Android platform in this work. Given that hardware variations can influence inter-arrival-time distributions, this feature was not considered in this study.

Bibliography

[1] Assigned Numbers | Bluetooth® Technology Website — bluetooth.com. https://www.bluetooth.com/specifications/assigned-numbers/. [Accessed 06-Jun-2023].

[2] BLE Advertising packet format | BLE Data packet format — rfwireless-world.com. https://www.rfwireless-world.com/Terminology/BLE-Advertising-and-Data-Packet-Format.html. [Accessed 06-Jun-2023].

[3] BluetoothxAE; Low Energy Packet Types - Developer Help — microchipdeveloper.com. https://microchipdeveloper.com/wireless:ble-link-layer-packet-types. [Accessed 06-Jun-2023].

[4] Core Specification Supplement | Bluetooth® Technology Website — bluetooth.com. https://www.bluetooth.com/specifications/specs/core-specification-supplement-9/. [Accessed 18-Jun-2023].

[5] The Bluetooth LE Security Study Guide | Bluetooth® Technology Website — bluetooth.com. https://www.bluetooth.com/bluetooth-resources/le-security-study-guide/. [Accessed 06-Jun-2023].

[6] Hidayet Aksu, A. Selcuk Uluagac, and Elizabeth S. Bentley. Identification of wearable devices with bluetooth. *IEEE Transactions on Sustainable Computing*, 6(2):221–230, 2021.

[7] Lei Bai, Lina Yao, Salil S Kanhere, Xianzhi Wang, and Zheng Yang. Automatic device classification from network traffic streams of internet of things. In *2018 IEEE 43rd conference on local computer networks (LCN)*, pages 1–9. IEEE, 2018.

[8] Johannes K Becker, David Li, and David Starobinski. Tracking anonymized bluetooth devices. *Proc. Priv. Enhancing Technol.*, 2019(3):50–65, 2019.

[9] Bruhadeshwar Bezawada, Maalvika Bachani, Jordan Peterson, Hossein Shirazi, Indrakshi Ray, and Indrajit Ray. Behavioral fingerprinting of iot devices. In *Proceedings of the 2018 workshop on attacks and solutions in hardware security*, pages 41–50, 2018.

[10] Louis Bienz. GitHub - LouisBienz/HomeScout: An Android App to scan, identify and classify BLE devices. — github.com. https://github.com/LouisBienz/HomeScout. [Accessed 06-Jun-2023].

[11] Robert Chojnacki. Item Tracker Sales Soar Alongside Luggage and Travel Accessories, Reports Circana — npd.com. https://www.npd.com/news/press-releases/2023/item-tracker-sales-soar-alongside-luggage-and-travel-accessories-reports-circana/. [Accessed 28-Jun-2023].

[12] Alexander Heinrich, Niklas Bittner, and Matthias 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*, pages 26–38, 2022.

[13] https://www.facebook.com/bbcnews. Apple AirTags - 'A perfect tool for stalking' — bbc.com. https://www.bbc.com/news/technology-60004257. [Accessed 06-Jun-2023].

[14] Pratibha Khandait, Neminath Hubballi, and Bodhisatwa Mazumdar. Iothunter: Iot network traffic classification using device specific keywords. *IET Networks*, 10(2):59–75, 2021.

[15] Jeremy Martin, Douglas Alpuche, Kristina Bodeman, Lamont Brown, Ellis Fenske, Lucas Foppe, Travis Mayberry, Erik C Rye, Brandon Sipes, and Sam Teplov. Handoff all your privacy: A review of apple's bluetooth low energy continuity protocol. *arXiv preprint arXiv:1904.10600*, 2019.

[16] Brett Molina. Apple AirTag trackers to receive privacy update amid stalking concerns — usatoday.com. https://www.usatoday.com/story/tech/2022/02/10/apple-airtags-privacy-update-stalking/6738071001/. [Accessed 06-Jun-2023].

[17] Hirofumi Noguchi, Misao Kataoka, and Yoji Yamato. Device identification based on communication analysis for the internet of things. *IEEE Access*, 7:52903–52912, 2019.

[18] Valentin Poirot, Oliver Harms, Hendric Martens, and Olaf Landsiedel. Blueseer: Ai-driven environment detection via ble scans. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 871–876, 2022.

[19] Faïçal Sawadogo, John Violos, Aroosa Hameed, and Aris Leivadeas. An unsupervised machine learning approach for iot device categorization. In *2022 IEEE International Mediterranean Conference on Communications and Networking (MeditCom)*, pages 25–30. IEEE, 2022.

Abbreviations

BLE	Bluetooth Low Energy
MAC	Media Access Control
SoC	System on a Chip
SVM	Support Vector Machine
MLP	Multi-layer Perceptron
API	Application Programming Interface
BER	Bit Error Rate
CRC	Cyclic Redundancy Check
PDU	Protocol Data Unit
UUID	Universally Unique Identifier
AD	Advertising
dbm	decibel-milliwatts
SIG	Special Interest Group
LSB	Least Significant Bit
RBF	Radial Basis Function
ReLU	Rectified Linear Unit
BR	Basic Rate
EDR	Enhanced Data Rate
IoT	Internet of Things
AI	Artificial Intelligence

Glossary

Regularization A technique in machine learning and statistics that applies a penalty to the complexity of the model. This helps in preventing overfitting by discouraging overly complex models which can fit the training data closely but perform poorly on unseen data. Regularization adds a penalty term to the original loss function of the model, with the aim of shrinking the parameter values towards zero (or a small value) to ensure simpler models.

Hyperparameter A parameter whose value is set before training a machine learning model. It is external to the model and cannot be estimated from the data. Examples include the learning rate, regularization strength, and number of hidden layers in neural networks.

Grid Search A method used to perform hyperparameter tuning in which a grid of hyperparameter values is exhaustively searched. For each combination of parameters, the model's performance is evaluated to find the optimal set of hyperparameters.

Dropout A regularization technique for neural networks where, during training, random subsets of neurons are dropped out (set to zero) at each iteration. This helps prevent overfitting by ensuring that no single neuron is overly reliant on its inputs.

Batch Normalization A technique to improve the training of deep neural networks. It normalizes the activations of the neurons in a layer for each training mini-batch, making the distributions of activations consistent and aiding in faster convergence and improved generalization.

BLE Device Any device which has BLE (Bluetooth Low Energy) capabilities, allowing for energy-efficient short-range wireless communication.

BLE Tracker A BLE device with the intended purpose of tracking and locating an item, often used for finding misplaced objects or ensuring the safety of valuable items.

Classic Bluetooth The original form of Bluetooth wireless technology, which provides a way for devices to connect and exchange data over short distances. It is distinct from BLE in that it is generally more power-intensive and is suited for continuous streaming of data.

List of Figures

List of Tables

Appendix A

Example link layer packets

In this chapter, example link layer packets will be shown to allow for further analysis and comparison.

A.1 AirTag

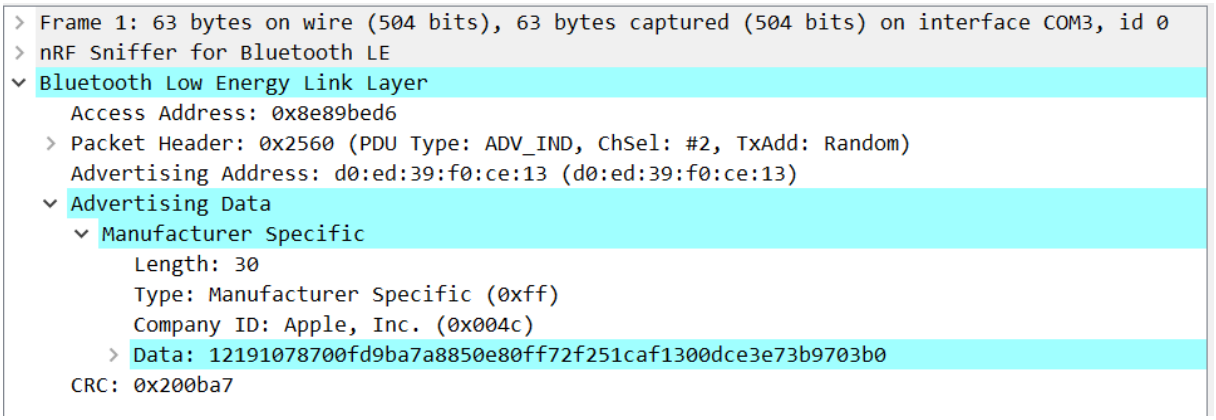


Figure 28: Interpreted information of example BLE link layer packets retrieved by Wireshark from AirTag.

Overview of example link layer packets collected from AirTag

Raw hex string	Total length (Bytes)	Number of AD	AD type list	AD length list
1eff4c0012191078 700fd9ba7a8850e 80ff72f251caf1300 dce3e73b9703b0	31	1	[Manufacturer Specific]	[31]

Figure 29: Attributes of example BLE link layer packets of AirTag.

A.2 Nutale

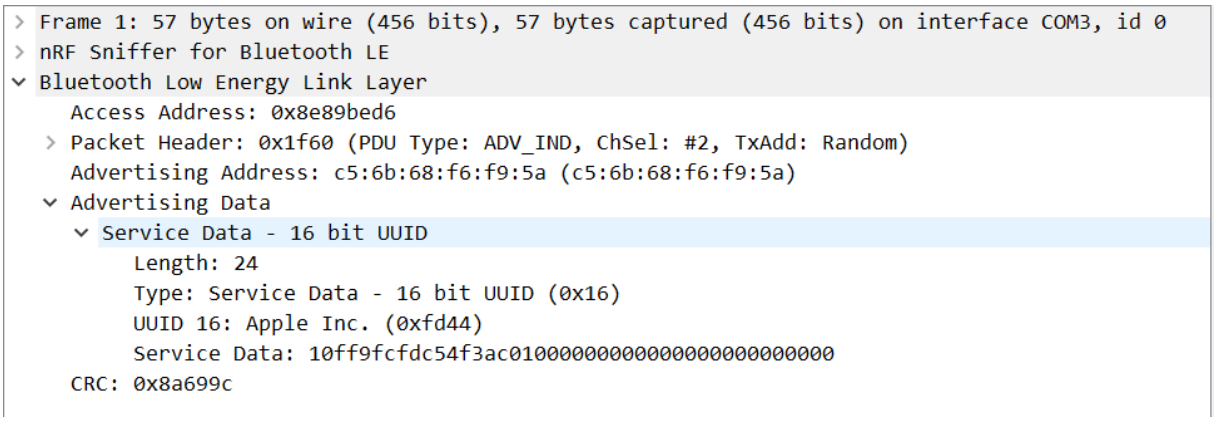


Figure 30: Interpreted information of example BLE link layer packets retrieved by Wireshark from Nutale.

Overview of example link layer packets collected from Nutale

Raw hex string	Total length (Bytes)	Number of AD	AD type list	AD length list
181644fd10ff9fcfdc54f3ac0100000000000000000000	25	1	[Service Data-16 bit UUIDs]	[24]

Figure 31: Attributes of example BLE link layer packets of Nutale.

A.3 HuaweiTag

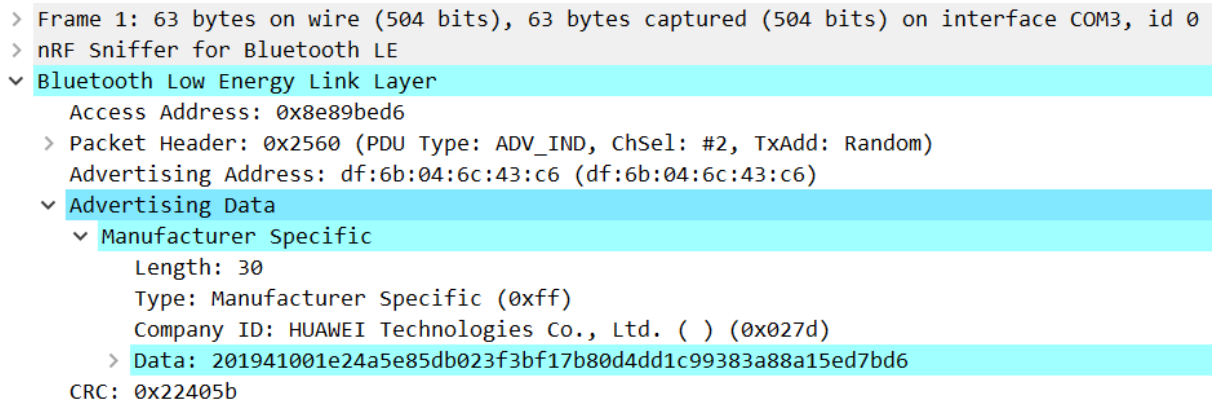


Figure 32: Interpreted information of example BLE link layer packets retrieved by Wireshark from HuaweiTag.

Overview of example link layer packets collected from Huawei Tag

Raw hex string	Total length (Bytes)	Number of AD	AD type list	AD length list
1eff7d02201941001 e24a5e85db023f3bf 17b80d4dd1c99383 a88a15ed7bd6	31	1	[Manufacturer Specific]	[31]

Figure 33: Attributes of example BLE link layer packets of HuaweiTag.

A.4 Tile

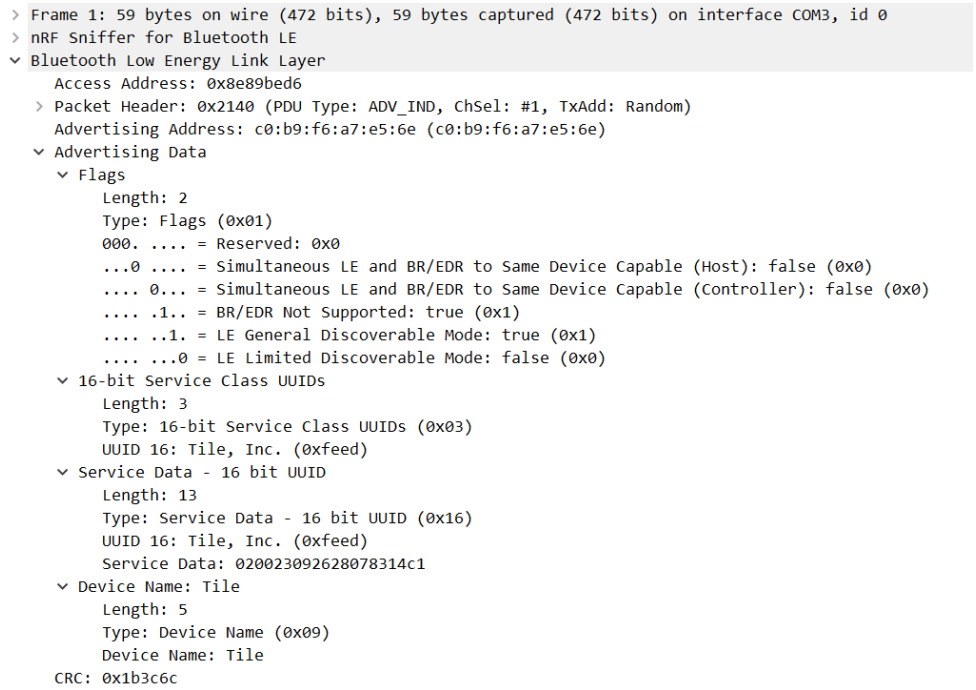


Figure 34: Interpreted information of example BLE link layer packets retrieved by Wireshark from Tile.

Overview of example link layer packets collected from Tile

Raw hex string	Total length (Bytes)	Number of AD	AD type list	AD length list
0201060303edfe0d16edfe020023092628078314c1050954696c65	27	4	[Flags, 16-bit Service Class UUIDs, Service Data-16 bit UUIDs, ,Device Name]	[3, 6, 14 ,6]

Figure 35: Attributes of example BLE link layer packets of Tile.

A.5 Three different states of an AirTag

Comparison of representative link layer packets of AirTag in three different states					
State	**Raw hex string**	**Total length (Bytes)**	**Number of AD**	**AD type list**	**Manufacturer Specific Data**
Unpaired	1eff4c000719050055100000017495ccea21dbcdf8ec128d79c4c1c6841018	31	1	[Manufacturer Specific]	071905005510000017495ccea21dbcdf8ec128d79c4c1c6841018
Nearby	07ff4c0012021402	8	1	[Manufacturer Specific]	12021402
Lost	1eff4c00121910787000fd9ba7a8850e80ff72f251caf1300dce3e73b9703b0	31	1	[Manufacturer Specific]	12191078700fd9ba7a8850e80ff72f251caf1300dce3e73b9703b0

Figure 36: Comparison of AirTag's patterns in three different states

A.6 Three different states of an HuaweiTag

Comparison of representative link layer packets of HuaweiTag in three different states					
State	**Raw hex string**	**Total length (Bytes)**	**Number of AD**	**AD type list**	**Manufacturer Specific Data**
Unpaired	1416eefd050300050c125258313204303011020b64	31	1	[Manufacturer Specific]	050300050c125258313204303011020b64
Nearby	0b16eefd0503000603070200	8	1	[Manufacturer Specific]	0503000603070200
Lost	1eff7d02201941001e24a5e85db023f3bf17b80d4dd1c99383a88a15ed7bd6	31	1	[Manufacturer Specific]	201941001e24a5e85db023f3bf17b80d4dd1c99383a88a15ed7bd6

Figure 37: Comparison of HuaweiTag's patterns in three different states

Appendix B

Device capture list

This chapter present a comprehensive overview of the specific BLE devices analyzed in this work. Link layer packets are captured from these devices to analyze if there are patterns in these devices for further classification purpose. The specifics of device capture list is presented in Table 1. The specifics of the devices capture list of unseen devices used in section 5.4 is shown in Table 2.

No.	Local Name	Device type	Number of collected frames
1	AirTag	Tracker	6516
2	HuaweiTag	Tracker	4644
3	Tile Slim	Tracker	4818
4	Nutale Air	Tracker	10165
5	Momax	Tracker	8362
6	Mili Tag	Tracker	1428
7	Baseus T2	Tracker	578
8	Baseus T2 Pro	Tracker	2577
9	Xiaomi Watch S1	Smart Watch	4481
10	Huawei Watch GT2	Smart Watch	3679
11	Oppo Smart Band 2	Smart Watch	4481
12	Edifier Lollipods	Headphone	2315
13	Edifier Headphone	Headphone	2216
14	Samsung Galaxy Buds Live	Headphone	4485
15	Huawei Freebuds 4E	Headphone	4833
16	Xiaomi Buds Pro	Headphone	1504
17	Haier Oximeter	Oximeter	5044
18	Konsung Oximeter	Oximeter	6767
19	Xiaomi Thermometer Pro	Thermometer	1982
20	Xiaomi Scale 2	Scale	4432
21	Huawei Scale 3	Scale	5303
22	Yuwell Glucose	Glucose Meter	5457
23	Omron isens631	Glucose Meter	4629
24	Omron u728t	Blood Pressure Meter	5229
25	Xiaomi Blood Pressure Meter	Blood Pressure Meter	3892
26	Lenovo Xiaoxin Laptop	Laptop	4195
27	iphone 13	Smart Phone	2842
28	iflytek Recorder	Recorder	3000
29	Dell Mouse	Mouse	3537
30	Samsung Galaxy Watch 3	Smart Watch	1504

Table 14: Specifics of device capture list

No.	Local Name	Device type	Number of collected frames
1	Huawei mangotek	Tracker	11897
2	Nut Focus	Tracker	1650
3	Huawei band 8	Smart Watch	4001
4	Logitech pebble mouse	Mouse	4753
5	Rapoo mouse	Mouse	5881
6	Ugreen mouse	Mouse	5593
7	Huawei scale Pro	Scale	5979
8	Xiaomi scale 8	Scale	8369
9	Yuwell blood pressure meter	Blood pressure meter	1259
10	Lepu Oximeter	Oximeter	1957

Table 15: Specifics of device capture list of unseen device