3-2021

Long Distance Bluetooth Low Energy Exploitation on a Wireless Attack Platform

Stephanie L. Long

LONG-DISTANCE BLUETOOTH LOW ENERGY EXPLOITATION ON A WIRELESS ATTACK PLATFORM

THESIS

Stephanie L. Long, Captain, USAF

AFIT-ENG-MS-21-M-058

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-21-M-058

LONG-DISTANCE BLUETOOTH LOW ENERGY EXPLOITATION ON A WIRELESS ATTACK PLATFORM

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Cyber Operations

Stephanie L. Long, B.S. Chemistry

Captain, USAF

March 25, 2021

AFIT-ENG-MS-21-M-058

LONG-DISTANCE BLUETOOTH LOW ENERGY EXPLOITATION ON A WIRELESS ATTACK PLATFORM

THESIS

Stephanie L. Long, B.S. Chemistry
Captain, USAF

Committee Membership:

Barry E. Mullins, Ph.D., P.E.
Chair

Timothy H. Lacey, Ph.D., CISSP
Member

Robert F. Mills, Ph.D.
Member

Stephen J. Dunlap, M.S., CISSP
Member

AFIT-ENG-MS-21-M-058

Abstract

In the past decade, embedded technology, known as the Internet of Things (IoT), has expanded for many uses. The smart home infrastructure has drastically grown to include networked refrigerators, lighting systems, speakers, televisions, watches, and more. The medical industry has developed IoT devices to monitor patient health, transmitting data wirelessly to phone applications that can be forwarded to a health professional. Networked sensors with wireless protocols are being used for military applications on the battlefield and to do triage in contested environments.

This increase in the use of wireless protocols provides a larger attack surface for cyber actors than ever before. Due to the nature of this wireless IoT communication, the data is susceptible to sniffing if the attacker is within physical proximity. While getting close and remaining undetected may be difficult for an attacker to do, directional antennas increase the range at which an attacker can sniff the data. Combined with the ability of medium-to-large sized drones to carry small, lightweight items, cyber attackers now have an option to simulate physical proximity by mounting an attack payload on a drone. Not only can the attacker collect data from a remote location, but she can also attack the networks and/or devices discreetly.

This research builds upon an attack sensor known as `skypie`, which was developed in previous research to do target geolocation, Wi-Fi beacon collection, and network attack [1, 2]. This study extends the cyber attack methodology for discreet wireless attack given these advances in technology. This prototype is designed using commercially available products, utilizes a directional antenna, and is lightweight and low-cost in order to be drone mountable and to emulate what a poorly funded, yet motivated threat-actor could produce. A new version of the `skypie` is developed

with Bluetooth Low Energy (BLE) collection capabilities for pattern-of-life analysis and attack capabilities to enumerate a device's characteristics and attempt to overwrite those with values of the attacker's choice. A distance of 600 meters is hypothesized to be the maximum distance at which the new package developed in this research, `BluBarry`, will be successful. To test the hypothesis, experiments where the `skypie` is conducting BLE focused Cyber Network Attacks (CNAs) are conducted at two prototype elevations. The goal is to determine the distances at which pattern-of-life data can be collected and at what distances the device can be interacted with by the attacker.

It is determined that the `skypie` v3 with the `BluBarry` package is able to passively collect BLE beacons at a Received Signal Strength Indicator (RSSI) sensitivity between -53 and -82 decibel-milliwatts (dBm) at an elevation of 3.05 meters (simulating drone height) from over a quarter mile away. One of the three BLE devices evaluated is able to be interacted with by the prototype up to a distance of 350 meters. Experiments are also conducted at a height of 1 meter to determine the effectiveness of the prototype on a drone at traditional street-level, conducive to attacks on ground-based platforms such as a rover, in a car, or simply sitting on a table. At this height, the `skypie` is able to collect BLE data from 350 meters and is able to connect to the device from 200 meters.

Acknowledgements

My first and most emphatic thank you goes to my Lord and Savior Jesus Christ, without Whom I could do nothing.

Thank you to my dedicated and knowledgeable advisor, Dr. Mullins. You have been patient and encouraging whilst advising me, and it is much appreciated.

Thank you to my committee members Dr. Lacey, Dr. Mills, and Mr. Dunlap for providing expertise, guidance, and advice on this wonderful thesis adventure.

I am grateful to my wonderful mother for encouraging me to pursue my degree and inspiring me to work hard, for providing snacks, and for being the all-around best mom.

Thank you to Clint Bramlette for all your kindness and advice.

Shout out to Arvin Bada for being helpful and somewhat funny.

Lastly, thank you to my dogs. They provided support by being the cutest, fluffiest pups.

Stephanie L. Long

Table of Contents

Page

Page

List of Figures

List of Tables

List of Acronyms

Abbreviation		Page
DoS	Denial of Service	25
ECDH	Elliptic Curve Diffie-Hellman	19
EDIV	Encrypted Diversifier	20
EDR	Enhanced Data Rate	8
FSPL	Free Space Path Loss	35
GAP	Generic Access Profile	18
GATT	Generic Attribute profile	12
GFSK	Gaussian Frequency Shift Keying	10
GPL	Ground Path Loss	36
GPS	Global Positioning System	38
GUI	Graphical User Interface	40
HCI	Host/Controller Interface	17
ICD	Implantable Cardioverter Defibrillator	25
ICS	Industrial Control Systems	24
IoT	Internet of Things	iv
IRK	Identity Resolving Key	20

Abbreviation Page

ISM Industrial, Scientific, and Medical 10

ISR Intelligence, Surveillance, and Reconnaissance 24

L2CAP Logical Link Control and Adaptation Protocol 18

LFSR Linear-Feedback Shift Register 31

LOS Line of Sight 36

LTK Long-Term Key 19

MAC Media Access Control 22

MD More Data 16

MIC Message Integrity Check 16

MIoT Military Internet of Things 23

MIoTL Mitigation of IoT Leakage 32

MITM Man-in-the-Middle 19

NACK No Acknowledgements 16

NESN Next Expected Sequence Number 16

NFC Near Field Communications 19

OOB Out of Band 19

Abbreviation		Page
PCs	Personal Computers	8
PDU	Protocol Data Unit	13
PHY	Physical	10
PINs	Personal Identification Numbers	26
RAND	Random Number	20
RSSI	Received Signal Strength Indicator	v
SCA	Sleep Clock Accuracy	15
SFTP	Secure File Transfer Protocol	39
SIG	Special Interest Group	8
SM	Security Manager	18
SN	Sequence Number	16
STK	Short-Term Key	19
SUT	System Under Test	60
TK	Temporary Key	19
UTC	Universal Time Code	52
UUID	Universally Unique Identifier	21

LONG-DISTANCE BLUETOOTH LOW ENERGY EXPLOITATION ON A WIRELESS ATTACK PLATFORM

I. Introduction

1.1 Overview and Background

The 1999 Disney movie *Smart House* was a fantasy movie that filled the minds of children with the possibilities of home automation fulfilling every snack desire and home activity [3]. The teenage boy in the family entered a contest and won a fully automated smart home where he could simply speak a request and his favourite milkshake would pop out of drawer in the kitchen, or he could request a channel change and the television would display his show of choice. Every gadget in the house was automated and connected to a hub (which turned out to be an evil computer) that controlled the house.

Even as far back as the 1950's, Westinghouse offered 16 different floor plans for what would be a home entirely controlled by electric power and automation [4]. Light sensors, automated laundry machines, sensors to display outdoor weather data on an indoor hub, and entertainment centers were all advertised features of these homes. While the electrified 1950's Westinghouse version of home automation faded into unrealized retrofuturism, even half a century later when Smart House was released, the 'smart home' was still firmly out of the grasp of the consumer. However, the year is now 2021, and home automation has quickly become a billion-dollar industry [5]. If manufacturers *can* connect a home appliance to the Internet, it seems like they will. In this haste, while society may enjoy the benefits of these products, it is vital

to consider, evaluate, and remediate the risks associated.

The expanse of Internet connected devices has greatly impacted everyday life in the 21st century. The technological advances in sensors and actuators have changed the landscape of how industries, hospitals, companies, and even homes operate. This ever-connected growing pile of embedded devices from appliances to wearables to literally the kitchen sink is known as the Internet of Things (IoT).

The idea of the Internet of Things began in 1982, when a group of programmers at Carnegie Melon University connected a drink machine to the ARPANET to monitor the supply and temperature of the machine [6]. This concept was initially known as an embedded Internet system. In 1994, an article published by the Echelon Corp posited the idea that data networks and control networks could work together to integrate embedded microprocessors in distributed networks [7]. This developed further in 1998 when a researcher from Stanford University discussed developments in embedded Internet technology where systems from handheld devices to factory automation and machine controllers could be networked [8]. Over the past two decades, IoT devices have been integrated into many different facets of life. Security Today reported 26.7 billion IoT devices currently in use, with an estimated 35 billion to be expected by 2021 and 75 billion by 2025 [9]. While the Internet of Things name may hint that the devices are solely based on TCP/IP protocols, there are several wireless protocols also intertwined under this definition to include: IEEE 802.11 (Wi-Fi), 802.15.4 (Zigbee), 802.15.1 (Bluetooth and Bluetooth Low Energy), and G.9959 (Z-Wave).

1.2 Problem Statement

Networked embedded devices are gaining in popularity for everyday use in homes around the world. Additionally, these systems are being used in hospitals, at the enterprise level, on the battlefield, and elsewhere. However, the lack of security,

particularly in devices utilizing the Bluetooth Low Energy (BLE) protocol, leaves users susceptible to data leakage and attacks. This research seeks to address the distance at which attacks can be made on BLE devices, using a lightweight, low-cost attack platform. Previous work on this platform has focused on use on a drone, and while this work does not preclude such usage, it continues without drone analysis. Work done by previous researchers such as Rose [10] and Beyer [11] have shown that important information can be gleaned passively on IoT and BLE devices, and can even be exploited from a long distance. This work furthers those bodies of research and that done by previous `skypie` researchers Bramlette [1] and Barker [2] to add BLE capabilities to an attack platform that can be controlled remotely and does not require the attacker in close proximity to the target.

1.3 Research Goals

The goal of this work is to further develop the `skypie` attack platform and `skyport` attacker interface to incorporate BLE capabilities. The previous prototype includes geolocation capabilities and Wi-Fi network exploitation. This work is relevant to the development of lightweight, low-cost attack platforms that are portable and drone mountable. However, with the proliferation of IoT devices, this research finds it especially advisable and beneficial to integrate pattern-of-life BLE collection into the `skypie` framework for furthering understanding of the risks presented by the growing footprint of the wireless IoT realm. The research seeks to answer the following questions:

- How close does an attacker need to be to collect BLE data from a target using lightweight, low-cost equipment?
- Can pattern-of-life data be collected at 600 meters?

- At what distance does pattern-of-life collection become infeasible?
- Can the attacker also interact with the target from that distance?
- How is the range of the `skypie` affected at a height of 1 meter versus 3.05 meters?

1.4 Hypothesis

This research hypothesizes that BLE pattern-of-life collection can be collected out to 600 meters using a cyber attack platform. This work aims to incorporate BLE collection capabilities into the `skypie` attack platform and determine the maximum distance at which an attacker could feasibly enumerate BLE version 4.0-4.2 devices. BLE beacons are collected from medical IoT devices at increasing distances, and connection to the device and an enumeration of current characteristics and values is attempted. The attacker is allowed remote control of the attack platform, with capabilities to attempt to overwrite characteristics on the BLE device. 600 meters is chosen as the theorized maximum distance due to the geolocation capabilities proven in the `skypie`'s first iteration by Bramlette. 600 meters is also roughly 6 times standard outdoor usages for Wi-Fi devices. While `skypie` v2 was proven to have Wi-Fi attack capabilities out of 2200 meters, BLE is a different technology than Wi-Fi, as BLE is known for its low energy, low powered, short-range capabilities, and thus 600 meters is still a lofty, but realistic goal.

1.5 Approach

A new `skypie` version is created to upgrade the attack platform to include BLE Cyber Network Attack (CNA) capabilities. The prototype consists of commercially available products with a Raspberry Pi 4 as the main brain. It utilizes a directional

antenna, and is supported by a Wi-Fi adapter, Bluetooth adapter, and microcontroller unit and GPS module for geolocation. This research maintains the original design goals to be lightweight in order to be drone mountable and low-cost to be achieved by a poorly funded threat-actor [1]. Furthermore, the `skyport` attacker interface is upgraded.

Medical IoT pulse oximeter devices utilizing BLE versions 4.0-4.2 are the focus of this research, but the platform can be used to exploit any version of BLE. While the goal was to collect up to 600 meters, the maximum range is found to be 450 meters. Data is acquired at collection points starting at 50 meters, increasing in 50 meter intervals to 450 meters for a total of nine points where beacons were collected. The experiments are run in an open field where BLE beacons are collected from each device, then connection and device enumeration are attempted. The attacker then attempts to overwrite values set on the device, depending on how the device has been configured by the manufacturer.

1.6 Assumptions and Limitations

This research and the experiments herein are bounded by the following assumptions and limitations:

1. All experimentation is conducted in an open, grassy, flat field. This choice minimizes the effects of buildings, trees, and other materials. However, as the experiments are run at different elevations, ground interference is a possibility.

2. The advertised distance for BLE v4.0 is up to 100 meters outdoors (assuming a powerful radio), and as this research utilizes a directional antenna and external Bluetooth adapter, experimentation begins at 50 meters for half the maximum distance, then gathers data in increments of 50 meters.

3. The location (optimal bearing) of the targets is assumed known.

4. BLE operates in the 2.4 GHz frequency, but as this experimentation is run in an open field, the interference of other devices in that 2.4 GHz range is considered negligible at least and indicative of what might be present in an urban environment at most.

1.7 Contributions

This research contributes to the body of knowledge on airborne wireless attack research, particularly in relation to BLE. It also analyzes the effectiveness of the attack platform at different heights to contribute to its use on different mediums or if the drone is flown at a lower elevation than normal. It also contributes to research on medical IoT BLE devices from both a pattern-of-life and enumeration/modification perspective.

It empirically proves that BLE pattern-of-life data can be collected at a distance of over a quarter a mile (450 meters), given a prototype height of 3.05 meters, and collection at approximately a fifth a mile (350 meters) with a height of 1 meter. This is up to 3.5-4.5 times the best-case scenario maximum BLE outdoor range of the devices [12].

1.8 Thesis Overview

The thesis is organized into six chapters. Chapter II gives an overview on the Internet of Things, then goes into what the BLE protocol is and how it works. The chapter continues on to discuss current research on BLE, vulnerabilities, attack vectors, and mitigation, then concludes with a description of the cyber attack chain. Chapter III describes the design of the `skypie` prototype developed in this research and the `skyport` interface. Chapter IV details the System Under Test (SUT) and

methodology used for the experiments. Chapter V discusses and analyzes the results of the experiments. Finally, Chapter VI summarizes the research and proposes future work for the research area.

II. Background and Literature Review

2.1 Overview

This chapter provides background understanding in Section 2.2 of what the Bluetooth Low Energy protocol is, how it is designed, and how it works. Section 2.3 continues with a description of related research in IoT, then on BLE and the vulnerabilities, attacks, and possible vulnerability mitigation techniques. The cyber attack chain is introduced in Section 2.4, followed by a discussion on signal propagation in Section 2.5. The chapter concludes with Section 2.6 where the remainder of this thesis is framed.

2.2 Bluetooth Low Energy

Bluetooth is a wireless medium that allows data exchange over short distances to build Personal Area Networks (PAN). The idea for Bluetooth originated in Sweden in 1989 when a researcher wanted to design wireless headsets [13]. It became a reality when the Bluetooth Special Interest Group (SIG) was launched in 1998, and in 1999 when the first Bluetooth Basic Rate (BR) mobile headset won the "Best of Show Technology Award" [14]. Bluetooth was integrated into Personal Computers (PCs) and mobile phones for the first time in 2000 [13]. The first computer it was integrated into was the IBM ThinkPad A30. Bluetooth v2.1 Enhanced Data Rate (EDR) was released in 2007 and incorporated Secure Simple Pairing. In 2009, Bluetooth v3.0 High Speed came out, and Bluetooth Low Energy (BLE) v4.0 followed a year later, with BLE v5.0 released in 2016. Once BLE was introduced, Bluetooth often was known as Bluetooth Classic (BTC), and BLE sometimes as Bluetooth Smart. While BLE bears the brand of Bluetooth, it was born out of the v4.0 specification, and the purpose and design goal are different from BTC.

Bluetooth was originally developed for short-range, constant transmission of audio, file transfers, or the use of a wireless keyboard. Such activities are typically time-intensive and thus require a sustainable power source. Conversely, BLE was designed to minimize power consumption, cost, and the data rate. In particular, it was intended for devices that only require periodic interaction with small data packet sizes. BLE v4.0 has a maximum range of 100 meters outdoors, with a more realistic indoor range of 10 meters [12]. For example, Bluetooth is the obvious choice for continuous streaming of music. A non power-intensive task, like interacting with home automation systems to turn a lightbulb on or off or get a temperature update, can be accomplished with BLE. The BLE protocol stack is divided into a controller, host controller interface, and host with the application layer on top (Figure 1) [15].

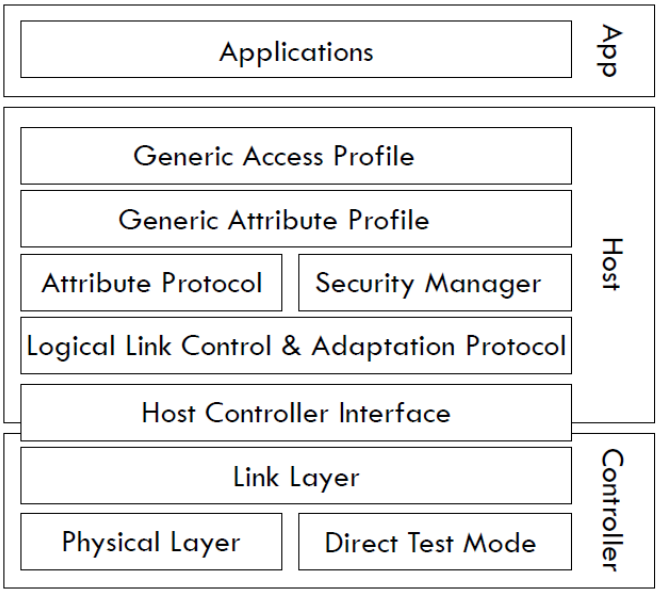


Figure 1: The BLE Protocol Stack (Adapted from [15])

2.2.1 BLE Controller

The controller is responsible for transmitting and receiving radio signals, and then interpreting the signals into data packets. The component consists of analog and

digital parts for the radio frequency and the hardware necessary to send and receive packets. The Physical (PHY) and Industrial, Scientific, and Medical (ISM) band operate over 40 radio frequency channels, each 2 MHz apart. Gaussian Frequency Shift Keying (GFSK) is used as the modulation scheme with a modulation index between 0.45 and 0.55, which allows reduced power consumption. BLE v4.0 has an advertised range maximum of up to 100 meters indoor/outdoor and BLE v5.0 has an advertised maximum range of 400 meters indoor/1,000 meters outdoor [12].

2.2.1.1 Link Layer

The LL is responsible for advertising, scanning, and establishing connections. As shown in Figure 2, there are 40 channels split into two types to accomplish this: advertising and data channels. Three channels, indexed as 37, 38, and 39, are used as advertising channels, and the other 37 are used as data channels [16].

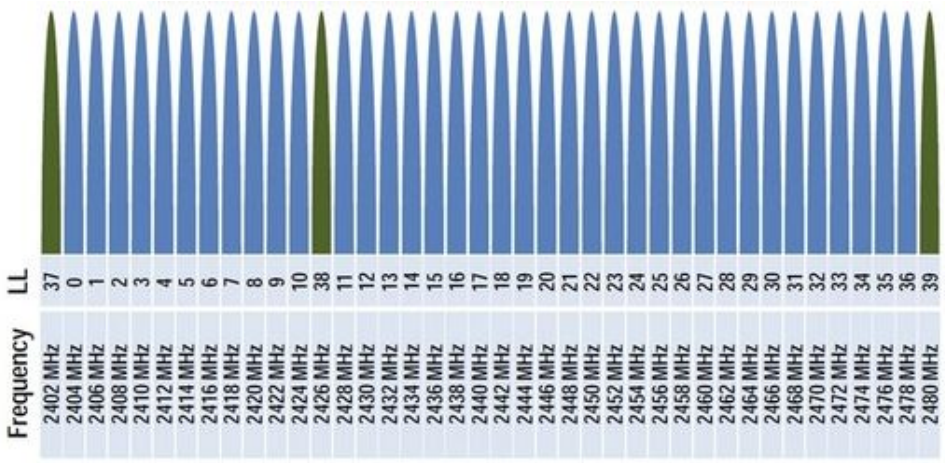


Figure 2: BLE Frequency Channels [16]

Since BLE uses the 2.4 GHz range, interference from other wireless mediums at the same frequency can be an issue. Wi-Fi, BTC, Zigbee, and other protocols also use this band, as well as commercial items such as microwave ovens and cordless phones. For this reason, the BLE system incorporated channel hopping, where the devices

only stay on a particular channel long enough to transmit and receive a single packet. Channel hopping uses Adaptive Frequency Hopping (AFH) to sense interference from other devices and hop across the channels accordingly to minimize such interference. Additionally, a unique access address is assigned to each connection as a correlation code for interacting devices. Figure 3 shows the five states that the LL can be in: standby, advertising, scanning, initiating, and connection.

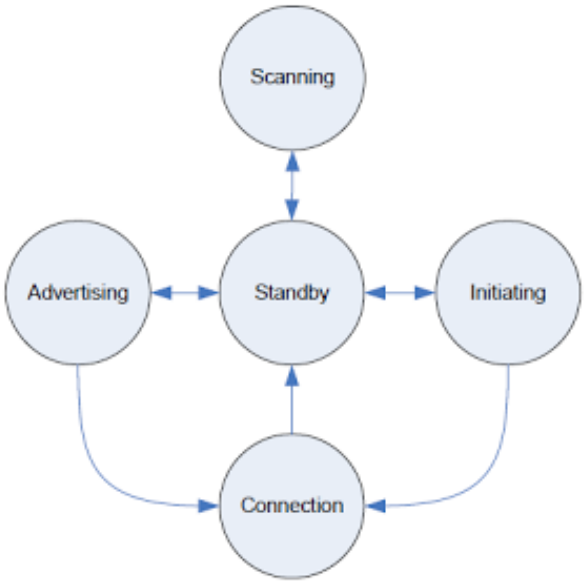


Figure 3: Link Layer States [17]

1. Standby: No packets are sent or received in the standby state. It can be entered from any previous state, and it is the default LL state.

2. Advertising: The advertising state, where the device is known as an advertiser, is responsible for transmitting advertising channel packets and responding to responses from the advertising channel packets.

3. Scanning: A scanner in the scanning state listens for advertisements.

4. Initiating: An initiator in the initiating state is responsible for responding to advertisement packets to initiate a connection.

5. Connection: The connection state is used to establish a connection between the two devices.

Once a connection is established, there are two roles for the devices. These roles have been referred to as master and slave, as well as central and peripheral. This research will use the central and peripheral terminology. The central, which is typically a phone, desktop/laptop, or tablet, determines the timing scheme for transmissions, and the peripheral, communicates information back to the central. As shown in Figure 4, a central device (c) can have up to seven active peripherals (p) in a piconet, which is an ad hoc wireless Bluetooth network that links two or more devices on the same physical channel. A scatternet is an ad hoc Bluetooth network consisting of two or more piconets. This allows for a device to act as a central to one device while simultaneously acting as the peripheral to another device in a separate piconet.

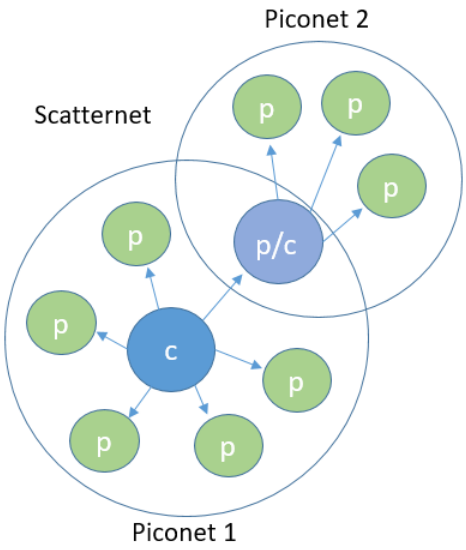


Figure 4: Scatternet Consisting of Two Piconets

A device in a connection at the link layer acts in one of two roles, either the central or the peripheral. At the Generic Attribute profile (GATT)/Attribute protocol (ATT) level, a device is either the central device (often a phone, laptop, PC, or another user

device) or the peripheral (embedded device). BLE terminology also refers to a server and client relationship. Generally, the user interactive device is the client, while the IoT device acts as the server and sends information as requested to the client. In this research, the terms central and client are used interchangeably to refer to a phone, laptop, or another device the user is interacting with. Similarly, the terms peripheral and server correlate to the sensor device.

2.2.1.2 Packet Types

Both the advertising channel packets and data channel packets have the same overall packet structure, but the payload differs, shown in Figure 5. The preamble is 1 byte long and has predefined values based upon whether it is an advertising or data channel packet. The next 4 bytes consist of the access address, which is always `0x8E89BED6` for advertising channel packets with a unique 4 byte value for a connection for data channel packets. The next field in the packet structure for BLE is called the Protocol Data Unit (PDU) and is variable in length from 2-39 bytes for an advertising PDU or 2-261 bytes for a data channel PDU. The packet is then appended by a 3 byte Cyclic Redundancy Check (CRC) to perform an integrity check for packet transmission.

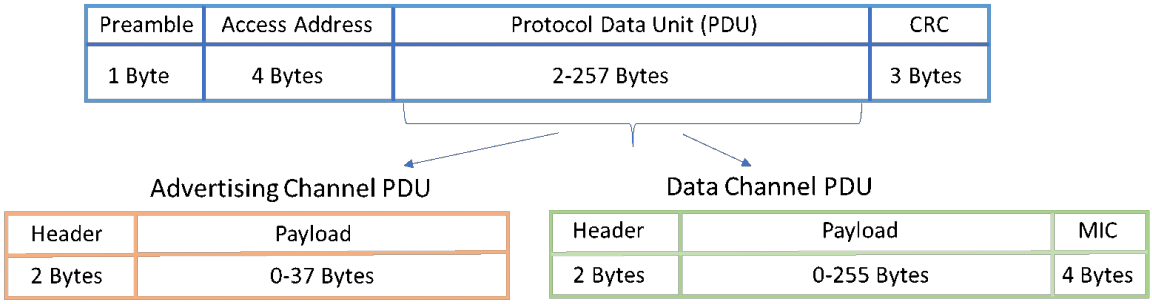


Figure 5: BLE Packet Structure (Adapted from [15])

The advertising channel PDU consists of a 2 byte header and a variable length payload. Within the header exists a 4 bit PDU type which is summarized in the Core

Specification for Bluetooth v4.2 [18]. The PDUs discussed in this research are:

- ADV_IND: Advertising indications, used by a peripheral device to advertise to any central device
- ADV_DIRECT_IND: Advertising direct indication, similar to the ADV_IND but targeting a specific central device
- ADV_NONCONN_IND: Advertising non-connectable indication, used to broadcast information to any listening device
- ADV_SCAN_IND: Advertising scan indication (ADV_SCAN_IND), which allows additional information to be requested by the central device
- SCAN_REQ: Request from the central requesting additional information from a peripheral
- SCAN_RSP: Response from the peripheral to the central following a SCAN_REQ
- CONN_REQ: Sent from the central to the peripheral to establish connection parameters [19]
- LL_TERMINATE_IND: Error code field sent to the peripheral from the central for why the connection will be terminated

The payload for the CONN_REQ contains vital data sent from the central (initiator) device to the peripheral (advertiser) to follow the connection [15]. When the connection is established between the central and peripheral, the central communicates the necessary data to sync the connection over various channels for the frequency hopping algorithm in the PDU. This allows for reduced power consumption by the peripheral as it can enter a sleeping period and wake up as specified by the central

to check for packets. Each packet transmitted during the connection between the central and peripheral is called a Connection Event (CE).

Figure 6 shows the fields encapsulated in the payload for the advertising PDU [15]. The Access Address (AA) connection value, which is unique to each BLE connection and is used as the source and destination address for the devices for the remainder of the connection, is the first field. The next slot is the random value used for the CRC Initialization calculation (CRC Init). The transmit Window Size (WinSize) and Offset (WinOffset) are used to determine the transmit window for the first CE between the devices. The subsequent CE timing schemes are determined by the connection Interval. The allowable peripheral Latency is next, then the connection supervision Timeout field, which is calculated based upon the connection interval and peripheral latency and is used to ensure that a connection is still ongoing. Connections can be dropped due to increased distance between the devices beyond the capable range, interference on the channels, or loss of power.

Advertising Channel PDU Payload									
AA 4 Bytes	CRCInit 3 Bytes	WinSize 1 Byte	WinOffset 2 Bytes	Interval 2 Bytes	Latency 2 Bytes	Timeout 2 Bytes	ChM 5 Bytes	Hop 5 Bits	SCA 3 Bits

Figure 6: Link Layer Advertising Channel Payload (Adapted from [15])

The Timeout monitors the connection state, and if inactivity is experienced greater than the agreed upon time, the connection state is exited and the peripheral returns to the standby state with a notification to the central, shown in Figure 3. The Channel Map (ChM) field indicates which channel index sequence the connection will use along with the Hop field to specify the hop pattern. The last field is the central's Sleep Clock Accuracy (SCA), setting the threshold for drift in clock synchronization between the devices [15].

The data channel PDU also consists of a 2 byte header, but is followed by a

variable length payload and an optional Message Integrity Check (MIC). The MIC is included only for encrypted LL traffic with a payload larger than zero. The header consists of an LL ID for whether the packet is a data or control PDU, the Next Expected Sequence Number (NESN), Sequence Number (SN), the More Data (MD) field, and the length of the payload and MIC to follow. The packet structure of an LL Control PDU includes a control field that sends requests, indicators, and responses to ensure the connection is still valid and to correspond with the central if the peripheral is getting data it does not understand or recognize.

The SN and NESN exchange is done in a manner as to ensure that data is not missed, working as Acknowledgements (ACK) and No Acknowledgements (NACK) [20]. For example, a central device sends an SN of 0 in the first packet and a NESN of 0. In order to convey an ACK on the part of the peripheral, the response packet should have an SN set to 0, with a NESN set to 1 to ACK the previous packet. If the NESN is set to 0, the central will retransmit the original packet until the peripheral replies with a SN of 0 and NESN of 1.

2.2.1.3 Lifecycle of a BLE Connection at the Link Layer

The lifecycle of a BLE connection can be tracked from the advertisement packet to the LL_TERMINATE_IND packet to end the connection [18]. Each connection spends a specified amount of time determined by the central device on a channel for each CE before hopping to the next channel. The Bluetooth SIG specifies that this connection interval be between 7.5 milliseconds and 4.0 seconds. If a peripheral has no information to send to the central, the specified time is still filled before hopping to the next channel. The following list details the interaction:

1. BLE device, such as a medical IoT device, advertises itself on one of the indexed channels 37, 38, and 39 at a time.

2. User device, such as a smart phone application, sends a CONN_REQ to the advertising device, specifying parameters for a synced connection.

3. Devices establish a connection; user device is the central and BLE device is the peripheral.

4. Devices hop to the next channel as specified by the CONN_REQ.

5. CE begins. The central sends a packet, known as the anchor, to the peripheral on the current frequency in order to confirm synchronization.

6. Peripheral responds if it has necessary data to transmit or its peripheral latency period is over. If no response necessary from the peripheral, the CE ends.

7. If the peripheral does transmit data, the CE continues as long as the central and peripheral have information to transmit within the specified data packet lengths and allowable time segment per hop. The incrementing sequence number embedded in the packet exchange serves as an acknowledgement. CE ends once either the packet length or time segment parameters are satisfied.

8. The connection remains active until the LL_TERMINATE_IND packet is sent by either the central or the peripheral [21].

2.2.2 Host/Controller Interface

The Host/Controller Interface (HCI) straddles the controller and host and allows the two components to communicate. Commands and data sent at the host level are translated by the HCI into relevant data to the controller and similarly translate events and data from the controller to the host.

2.2.3 Host

The BLE host consists of the Logical Link Control and Adaptation Protocol (L2CAP), Security Manager (SM), the Attribute protocol (ATT), the Generic Attribute profile (GATT), and the Generic Access Profile (GAP) [15].

2.2.3.1 Logical Link Control and Adaptation Protocol and Security Manager

The primary objective of the L2CAP is to multiplex data for up to three logical channels. It is also responsible for segmentation and reassembly of packets that may exceed the single packet length allowable. An L2CAP channel is a single bidirectional data channel used by a particular protocol. Channels in this circumstance are not the RF channels previously discussed, but rather "channel" refers to "a single sequence of packets, from and to a single pair of services on a single device" [15]. BLE uses three fixed channels with channel IDs of: `0x004` for ATT, `0x005` for the LE signaling channel, and `0x006` for SM. The SM is responsible for low-level security via pairing, bonding, key distribution, encryption, and signing.

2.2.3.2 Pairing and Bonding

In general, the pairing process at the host level happens in three phases. The first phase begins after the connection request has been sent with the necessary information for the two devices to stay synced and the LL connection has been established. The devices share capabilities and requirements for authentication and bonding. The central device sends a pairing request and receives a pairing response. The devices now enter phase two where pairing is completed over the SM with legacy or secure connections pairing. However, not all devices initiate secure pairings, as this is an option within the BLE specifications [18]. A peripheral device may be connectable

by the central without proceeding to pairing or bonding [22]. In legacy pairing, the Temporary Key (TK) is used along with a random number from both the central and peripheral to generate an Advanced Encryption Standard (AES) 128 bit encrypted key known as the Short-Term Key (STK). There are three different methods that devices could agree upon to complete the initial exchange to the TK: Just Works, Out of Band (OOB), and a passkey.

1. In Just Works, the TK is predefined as zero
2. The OOB method requires the TK to be exchanged over a medium other than BLE, such as Near Field Communications (NFC)
3. A passkey is a six digit number set by a user and transferred between the devices

After the TK and random numbers are used to generate the STK, the STK encrypts the communication using AES.

For BLE devices using v4.2 and on, devices may choose secure connections pairing. Secure connections pairing does not use a temporary or short-term key, but rather uses Elliptic Curve Diffie-Hellman (ECDH) public and private key pair infrastructure to compute a Diffie-Hellman key, which is then used to authenticate the connection and enter into phase three where the Long-Term Key (LTK) is generated, and the connection is encrypted. There are four pairing methods with secure connections:

1. Just Works: In the Just Works technique, both systems exchange public keys and then generate a nonce for each device to calculate a confirmation value. A matching confirmation value allows the connection to continue.
2. Numeric Comparison: Numeric comparison is accomplished similar to Just Works, except a step is added to combat Man-in-the-Middle (MITM) attacks. After key exchanges, each device generates a six digit value using a nonce, and the user manually checks the values to ensure they match.

3. Out Of Band: OOB pairing requires the public keys, nonces, and confirmation values for each device to be sent over another technology such as NFC.

4. Passkey: The passkey pairing method utilizes a six digit number input by a user, the public keys for the central and peripheral devices, and a 128 bit nonce. Once the values have been verified, the connection can proceed to phase three. Although no communications up to the connection authentication at the end of the pairing methods have been encrypted, the complexity of these values better protects against MITM and other sniffing attacks than the legacy pairing methods.

Bonding is optional and occurs in phase three where the LTK is created. This key is stored in a security database on both devices and maintained so the pairing process does not have to be repeated with each connection. Other keys and values are also generated and distributed in phase three. The Encrypted Diversifier (EDIV) and Random Number (RAND) values are used to generate and identify the LTK, the Connection Signature Resolving Key (CSRK) is used to sign and verify transmitted data for authentication, and the Identity Resolving Key (IRK) is responsible for resolving private addressing so peer devices can reveal identities of the other.

2.2.3.3 Attribute Profiles

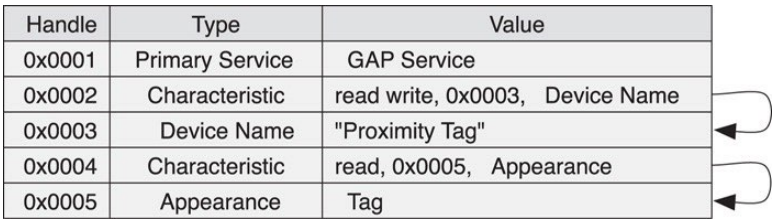
The ATT exists on top of the L2CAP layer and sets the parameters for how data is accessed on the server by the client. Data is stored on the attribute server in collections of "attributes" that are available for read and write by the client and is managed by the GATT. Attributes are the data entities that consist of necessary user data that are organized by the GATT server. An attribute, as seen in Figure 7, an attribute consists of a handle, a type, permissions, and a value:

1. Handle: a 16 bit identifier unique to and addressable for each attribute

2. Type: a Universally Unique Identifier (UUID) that identifies the type of data present in an attribute, whether that be a characteristic, service, or another type such as a heart rate measurement, temperature, or another vendor-specific UUID

3. Permissions: specifies whether an attribute can be read or written to, the security levels, and if it can be notified or indicated

4. Value: the actual data of the attribute



Handle	Type	Value
0x0001	Primary Service	GAP Service
0x0002	Characteristic	read write, 0x0003, Device Name
0x0003	Device Name	"Proximity Tag"
0x0004	Characteristic	read, 0x0005, Appearance
0x0005	Appearance	Tag

Figure 7: Attribute Example [15]

The GATT dictates how profile and user data are transferred over a BLE connection. It defines the format of services, characteristics, and descriptors, and the procedures to access the attributes between a server and a client. A service consists of one or more attributes that holds characteristics. A characteristic holds a particular value such as a current heart rate, how bright a lightbulb might be, the current battery life of a device, etc. A characteristic consists of three parts:

1. Declaration: assigned a value for the start of a characteristic and can be used to group the attributes for a particular characteristic

2. Value: actual value for the characteristic

3. Descriptor: additional data or configuration for the characteristic such as the characteristic extended properties, user description, presentation format, ag-

gregation format, client characteristic configuration, and server characteristic configuration

2.2.3.4 Generic Access Profile

GAP defines device discovery and device connections [15]. There are two pairs of roles available:

1. Broadcaster and Observer: these roles are where one device is broadcasting information about itself, such as name, Media Access Control (MAC) address, TxPower, flags, etc., and the other device observes that information for data collection. The devices are not connected in these roles.

2. Central and Peripheral: these roles have been previously discussed in Section 2.2.1.1 and are involved in a connection.

GAP defines the discoverability of a device to be:

- Non-discoverable: this is the default mode for a device and must be changed by the host. A device with this set means scanning devices should ignore it [15].

- Limited-discoverable: when configured to be limited-discoverable, a device will advertise its information at a rate between 250 - 500 milliseconds, and will appear at the top of a list during a scan. Typically, a device will be in limited-discovery mode for the first 30 seconds after being turned on, then defaults to general-discoverable if no connection is made. This is to allow other limited-discoverable devices to be found.

- General-discoverable: a device in general-discoverable is discoverable but has not been interacted with recently. It advertises every 1.28 - 2.56 seconds, and will appear on a scan after the limited-discoverable devices.

Devices are also filtered by TxPower, which is the transmitted power of the packet in decibel-milliwatts (dBm), to determine which devices are likely are closer in proximity. The devices are also filtered by types of services offered to determine which devices are typically interacted with more based on the services a device supports. Once a connection is made between the central (i.e., a user phone application) and peripheral (BLE device), the central may perform an enumeration of all services and characteristics of a device, or it may only read a particular service that it is more interested in at that time. Additionally, GAP defines connection modes, bonding, and the security modes used during pairing.

2.3 Related Research

2.3.1 Academic Interest in IoT

The Internet of Things has been a hot topic for academia due to its many applications. Research spans embedded devices, technological advancements, cyber attack, vulnerability mitigations, and risk frameworks. Within these areas exist military edge advancement, industrial uses, home automation systems, agricultural applications, and medical devices.

Recent research has been published about integrating IoT technology on the battlefield to help gain a strategic advantage over the enemy and to keep situational awareness on an unfamiliar terrain. Yushi et al. studied the application of the Military Internet of Things (MIoT) in 2012, examining the uses of IoT in the battlefield as information sensing, information transmission, and information serving [23]. The authors discussed the advantage that IoT can bring to shorten decision making times, increase information sharing, and enhance the capabilities of a system to include multiple connected systems. The research presented a model for a MIoT architecture consisting of a base sensing layer where all nodes collect data on the battlefield, which

is then sent to the access layer to collect and coordinate the data from the various wireless IoT protocols used. From there, the model proposes a network layer to transmit to the LAN, then to a service layer for data storage, and finally an application layer for Command and Control (C2), Intelligence, Surveillance, and Reconnaissance (ISR), and decision making. The devices in a MIoT environment would have three modes: the first for physical sensing and data collection, the second for network transmission to combine the data, and the third for information serving for human interaction. This research was provided a proposed architecture and application modes for the MIoT.

The BATDOK project is developed by the military as a trauma kit for the battlefield [24]. The application has the ability to triage patients and collect patient trends, all with the use of medical IoT devices. Research by Johnsen et al. provided a perspective on the benefits IoT could bring in military operations during a disaster [25]. The paper demonstrated how networked devices can be used to bring situational awareness in real-time that otherwise may not be possible. The employment of IoT devices in this scenario allowed for enhanced planning, monitoring of health and capabilities, and better augmentation of military forces for information collection and processing.

Sadeghi et al. researched the security and privacy challenges associated with the IoT in an industrial setting [26]. Industrial Control Systems (ICS) are networked to provide for flexible and more efficient organization and management of industrial systems. With the increased capabilities that IoT offers to ICS comes increased security risk for cyber attack. The paper discussed various cyber attacks on ICS, to include Stuxnet, which provided a dangerous example of how exploitation of ICS can result in physical consequences. The difficulties in securing industrial IoT devices is also discussed as many of these systems have physical limitations such as constrained

computational, memory, and energy resources, and are not afforded restarts or down-time that updates and patches may require. The importance of integrity verification of the software on the systems is highlighted.

The IoT has been incorporated into everyday life in homes from lighting to the kitchen to security cameras for property surveillance, babies, and pets, to speakers and watches, and more. Geneiatakis et al. shared the security and privacy issues associated with an IoT smart home [27]. The authors discussed different types of IoT-related attacks: a potential for pattern-of-life analysis through eavesdropping, active impersonation of devices in the environment, software exploitation, or even a Denial of Service (DoS) attack.

The ability to transmit the data back to a doctor to allow better response for patients has gleaned much interest in the academic community, with more than 2,400 IEEE articles published with the keywords "medical IoT" since 2015. As early as 2008, Halperin et al. analyzed an Implantable Cardioverter Defibrillator (ICD) that communicated patient health data wirelessly [28]. The team determined that the medical information, such as the patient's name, date of birth and cardiac values, was sent unencrypted and could be intercepted by a malicious actor. Furthermore, the researchers were able to exploit a testing interface to replay communication to the ICD in order to induce a shock to the patient's heart. This attack could result in death to a victim of the man-in-the-middle attack, and the device had no security measures in place to stop such an attack. Rahman et al. analyzed the poor security design utilized in the Fitbit (version unspecified) that allowed an attacker to reverse engineer the protocol and inject false fitness data to the online tracker [29]. One of the security flaws the researchers discovered was that the network traffic showed user credentials were passed across the network in cleartext upon authenticating with the software. Furthermore, the data for the log files were also sent unencrypted, giving

access to the actual health data. The researchers then analyzed the possible attacks on the system and developed an attack framework against the device.

Wang et al. discovered wrist-wearable BLE IoT devices, such as fitness monitors, could be used to extrapolate banking Personal Identification Numbers (PINs) [30]. The researchers were able to determine the PINs based on two methods: sniffing attacks and an internal attack by putting malware on the user's app device. Data collected from the accelerometer was directly mapped to the key pushed on an Automatic Teller Machine (ATM) machine or a door entry keypad. The data pulled from the devices and the development of a distance-estimation algorithm allowed the attackers to determine the user's PIN with an 80% accuracy upon a one-time key entry and 90% upon a three-time key entry.

2.3.2 BLE Research

Current research on BLE shows several successful attack vectors. One of the most well-known BLE hacking landmark papers was published by Mike Ryan on hacking BLE v4.0 [31]. He demonstrated how packet transfer at the link layer could be passively collected using a Bluetooth sniffer such as the Ubertooth One. His research determines the four values unique to connection–hop interval, hop increment, access address, and CRC Init. Ryan highlights the cleartext information that can be found, such as the actual passkey and privilege escalation passwords. If the devices are utilizing legacy encryption methods, a tool presented in the research called `Crackle` can be used to brute force the temporary key based on a six digit pin and extract the LTK. The creators of BLE attempted to protect the protocol from cyber attacks such as sniffing or MITM attacks by requiring the nonce exchanged at the beginning of a session and the need to only exchange the LTK once between devices to ensure subsequent connections are re-established and remain encrypted without sending the

key again [18]. Ryan's research determined attacks that can be used to force these values to be resent. The session nonce is set at the beginning of a connection, so the attacker only needs to jam the current connection to force a reconnect to capture the session key. To ensure the LTK is exchanged, the eavesdropper can force a key renegotiation by injecting an LL_REJECT_IND at the proper time, resulting in a new connection.

Gutierrez del Arroyo's 2016 DEFCON presentation outlined a process for hacking BLE using a sniffer to gather cleartext data from a thermostat [32]. The thermostat the researcher cracked is the same model that an art museum boasted of using to secure the environment for a painting. Using the data collected, he was able to gather the password used for the device, the firmware binary for updates, and replay crafted packets to the device to change these characteristics. He was also able to impersonate the app interacting with the device and even force a reset for the entire system.

Rose et al. highlighted the absence of a pre-existing tool to accurately determine Bluetooth and BLE device distance [33]. While the `Blue Hydra` tool may be able to locate devices at a short range, the `BlueFinder` tool established in this work showed improved accuracy at long-range distances over 50 meters, adapting the long-distance path loss model utilized in a Zigbee range tool, `zbdfind`. The development of this tool allowed the researchers to hack a BLE lock from over a quarter mile away.

Rose [10] and Beyer [11] determined that pattern-of-life behavior can be gleaned from passively sniffing BLE connections and acting as a MITM. Even if the attacker does not pursue stealing the communication, she can determine what time someone is present, who might be in the house based upon the devices used, and correlate other data to determine a person's route and routines. This could be used by a malicious actor to determine the best time for a cyber attack, may enable a burglary, or even

could be used to locate a particular target.

A new Common Vulnerabilities and Exposures (CVE) was published in 2018 under the ID CVE-2018-5283 that disclosed that both BTC and BLE devices utilizing ECDH key pairs were still susceptible to MITM attacks [34]. The Israel Institute of Technology discovered an Invalid Curve Attack on the ECDH algorithm to recover the session key of an encrypted connection [35]. While BLE secure connections featured public key authentication, its use was not required. This oversight in its functionality allowed manufacturers to choose not to use it, whether on purpose or by ignorance. Without public key authentication, the pairing mechanism is once again susceptible to passive sniffing and a MITM. The BLE SIG has since updated BLE specifications in v4.2 to require key validation [18]. Devices are susceptible to this attack until device manufacturers develop and make available an update, which may require interaction from the user. It should be noted that it is unlikely for a user to seek out an update as IoT devices are typically meant for one purpose and researching a security update for the device is probably not a priority for the user.

In 2019, the BLE-enabled Xiaomi M365 Electric Scooter, used in many ride sharing apps, was proven to be hackable by a Dallas company called Zimperium [36]. The scooter had a companion app that allowed users to use features such as an anti-theft system, cruise control, eco mode, and an option to update the firmware. The company determined that although this app required a password, authentication had not been properly set up to verify on both the app and the device itself. This allowed the attackers to sniff the data upon initial connection to the scooter, determine what payload looked like and sequence numbers, and were able to craft their own payload for malicious purposes up to 100 meters away. The commands leveraged by the company showed the ability to commit a DoS where the scooter enters a locked state and the user is unable to override via the scooter. There is also the ability to cause a rider

to accelerate or break suddenly, putting their life in potential danger. Zimperium also demonstrated that new firmware could be flashed to the scooter to take it over completely or to destroy its intended purpose.

Booth et al. were able to replicate Zimperium's results and detail the process [37]. The researchers used the `bettercap` tool for recon to enumerate the device of interest. They wanted to determine the results that each characteristic and service had on the scooter, and noted that this could have been completed by sniffing the traffic between the app and the device, by analyzing the binary for the app software, or by analyzing the firmware on the scooter. Since they had procured their own scooter, they checked the firmware and discovered it was unencrypted. They analyzed the BLE characteristics by using the gattacker tool as a MITM to trigger various demands and determine the characteristic values each action mapped to.

Long et al. examined the Masimo pulse oximeter used as one of the targets in this research to show how the attack surface for malicious actors has increased as IoT gains in popularity, putting users' privacy at risk [38]. The researchers utilize an Ubertooth sniffer [39] to passively sniff the data being transferred between the user phone application and Masimo device in cleartext. The manufacturers of the Masimo device attempted to obfuscate the data in the packets, but through the use of the Ubertooth, the `hci_snoop` log from the phone, and by doing static and dynamic analysis of the phone application, the researchers were able to determine what portions of the packets contained the health data.

2.3.3 BLE Vulnerabilities and Attack Vectors

BLE has some security measures built into its design, such as the use of connection-specific access addresses, channel hopping, the use of data whitening, CRCs at the end of each packet, communication encryption with 128 bit AES, varying pairing

methods, and the use of ECDH algorithms for v4.2 and later key generation. A unique access address is assigned to the connection between two devices, and the interactions between the devices is obscured and hard to track if the initial assignment of the access address is not captured. Channel hopping provides security for BLE connections since the hopping sequence is unique to each connection. However, this information can be sniffed in the initial CONN_REQ or calculated based upon the time it takes to return to a channel, then dividing the time across the 37 channels [31]. Data whitening is the practice of scrambling data based on the last six digits of the device clocks in order to make the data difficult to follow. Nevertheless, this value can be known, and the whitening sequence can be found by simply XORing the data with that value.

Another security measure within BLE is the use of the CRC checksum. The CRC must be verified at the end of every packet, and the calculation necessary to do so is dependent on the CRC Init value, which is sent in the initial CONN_REQ. A connection is encrypted following the pairing process, of which there are different kinds, and optional ECDH algorithms are used depending on pairing scheme, as detailed previously.

BLE is susceptible to several different types of attacks. The most prevalent are passive eavesdropping and a MITM attack. Passive eavesdropping can be accomplished by a third-party entity monitoring the traffic transmissions between devices. If this information is sent in cleartext, credentials and command text can be easily gleaned. Even if the data is encrypted, a clever attacker can use clues present in the packets to crack the key, dependent on the encryption scheme. A MITM attack occurs when passive eavesdropping is taken a step further and the attacker intercepts the data between the devices and acts as a proxy, making herself a middle-man and posing as the peripheral device to the central and vice versa. The attacker can inject malicious code instead of forwarding the intended data from the peer device.

It is important to note that some devices connect but do not require any pairing, such as the PlayBulb lightbulb manufactured by MiPow. Since no pairing is involved, all communications are sent in cleartext and can be sniffed, and data can be collected, analyzed, and even duplicated to send commands by an attacker. While BLE connections are difficult to interrupt once established, the initial pairing of the devices is vulnerable. Because there is no encryption at the link layer before a connection is made, a malicious actor or curious bystander could use a third device to sniff the data being exchanged between two devices. This vulnerability is particularly dangerous when the connection request and subsequent pairing happen for the first time or after one of the devices have forgotten the agreed upon parameters.

According to Mike Ryan's research in 2013, BLE v4.0 could be cracked once four main values are sniffed: the access address, the hop interval, hop increment, and CRC Init [31]. All of these values can be observed and decoded. The connection-unique access address is available in the link layer data of the CONN_REQ. The hop interval can also be found there, and it can be discerned by observing how long of a distance between two consecutive packets. The hop increment can be found by determining when the time packets arrive from one channel to the next. CRC Init requires the Linear-Feedback Shift Register (LFSR) with the CRC which can be pulled from a packet and reversed. The CRC Init value will then be the value left in the LFSR.

2.3.4 BLE Vulnerability Mitigation

BLE attacks center around either the lack of connection encryption or the vulnerabilities present in the initial key exchange to provide communication encryption. The devices are susceptible to passive sniffing, device spoofing, and MITM attacks. The research summarized above on legacy pairing demonstrated a need for a secure pairing methodology between devices, as well as the need for stronger authentication,

especially in firmware update mechanisms. These discovered vulnerabilities resulted in the secure connections pairing methodology in later versions which utilized ECDH key pairing. However, it is the responsibility of IoT system manufacturers to ensure they are implementing the most recent versions of BLE and enabling the security features on their devices.

Other research to mitigate intrusions include anomaly detection tools to determine when an intruder is interrupting the connection or posing as another device. Sniffers can be used to collect data on connections for audit purposes. Although current BLE sniffers can only follow one connection at a time, the `BLE-Multi` tool can simultaneously capture multiple active connections which can be used for auditing [40].

Another tool is `BlueID`, which was developed as a mitigation against Bluetooth device-spoofing attacks. An attacker can spoof many things for a device such as the MAC address and name, and can even craft packets to display certain time frames and sequence numbers. Research on the use of `BlueID` demonstrated that fingerprinting a device based on the timestamp of the temporal feature of frequency hopping cannot be spoofed [41]. While this tool was initially targeted for BTC, it has been recently updated to include BLE. Lastly, another mitigation is needed for the pattern-of-life vulnerabilities presented with passive sniffing. Even if traffic encryption is used, a device still broadcasts its MAC which can be used to infer what type of device is in the environment and even correlate who the user is, or whether someone is home. Beyer developed the `Mitigation of IoT Leakage (MIoTL)` tool in his research to provide spoofed IoT device traffic to make it difficult for an attacker to determine any sort of pattern-of-life analysis [11]. While the benefit from such a tool would be tremendous, the interaction and setup required on the behalf of the user makes this current solution less feasible for the typical home.

2.4 Cyber Attack Chain

Cyber attack is generally done in a specific, methodical order to maximize effects and reduce detection. While the naming conventions may differ across sources for the methodology, the overall approach is the same. Figure 8 shows these steps: reconnaissance, scanning, access and escalation, exfiltration, sustainment, assault, and obfuscation [42].

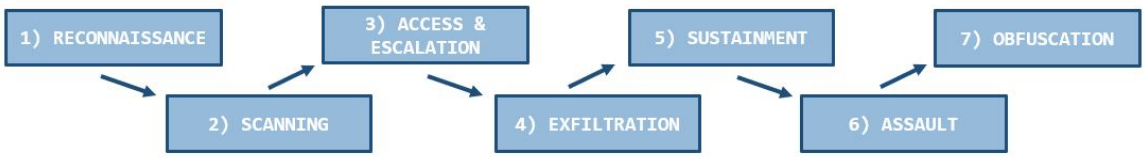


Figure 8: Cyber Attack Chain

To begin the cyber attack process, the first step is reconnaissance. This step is done before any attacks, as the available targets must be identified. At this point, all work by the attacker is passive, and the attacker does not interact specifically with a target. Examples of recon include phishing, social engineering, dumpster diving, etc. The goal of this step is to identify what targets may be of interest to attack. The next step in the chain is scanning, or active reconnaissance. Here is where an attacker may interact with a victim to determine where there may be vulnerabilities, and how they will be able to interact with or exploit a target. Examples of this include `nmap` scans [43] with Wi-Fi or scan requests with BLE. This allows the attacker to identify specific details about a network or device. In some methodologies, steps one and two are combined [44]. Step three is to gain access and escalate privileges. This step requires an attacker to develop the most appropriate exploit for a particular system of interest using the results of reconnaissance and scanning to increase the probability of success. Password attacks, relay attacks, or MITM attacks are some methods that may result in access. Once on a host, privilege escalation allows for the attacker to

have the highest level of authorization of the device. Traditionally, on computers running Windows this is "system" access, and on Linux, the famed "root" access. On embedded systems, this typically refers to having unrestricted access to all data and controls. Any of these system environments could be equipped with BLE technology.

Step four is to exfiltrate any pertinent data on the target device, such as passwords, banking information, proprietary data, etc. Ensuring that access is maintained by installing a listener, beacon, backdoor, or rootkit that allows the attacker to revisit the victim unobstructed is the fifth step. The sixth step, assault, is optional and not used in every attack, because the goal is to destroy some functionality of the victim device, such as "bricking" the device so it is destroyed for the user, such as with Silex [45]. The final step in the cyber attack chain is obfuscation to cover tracks. The attacker typically desires to be able to return to the victim system later, which usually involves removing evidence of intrusion so that the owner is unaware the target has been compromised. To ensure this, she may alter or delete logs, remove command history, or remove added software, for example.

2.5 Signal Propagation

The BLE technology travels wirelessly in the 2.4 GHz range. The signals propagate through space from a transmitter to a receiver. The TxPower, briefly discussed in 2.2.3.4, is advertised from the transmitter to the receiver to allow for a Received Signal Strength Indicator (RSSI), an estimate of the radio strength, to be calculated. The exact RSSI value may have variation, even at a fixed distance due to the chipset manufacturing and interference from outside systems [46]. However, the trend of average RSSIs collected is useful information for distance estimation - the height the RSSI value, the closer the object is estimated to be in the absence of obstacles. The LE specification requires that TxPower be no weaker than -20 dBm and no stronger

than 10 dBm, corresponding to a minimum of 10 µW and a maxmimum of 10 mW, respectively [18]. A general expression to determine the expected RSSI is

$$P_{rx} = P_{tx} + G_{tx} + G_{rx} - L_p \tag{1}$$

where P_{rx} is the expected RSSI, P_{tx} is the transmit power of the BLE device, G_{tx} is the dBi gain of the same, G_{rx} is the receiving antenna dBi gain, and L_p is the path loss measured in dB. Figure 9 provides a visual for this equation. Path loss is defined as "a measure of how much the radio signal has reduced in power between the antenna in the transmitter and the antenna in the receiver" [15].

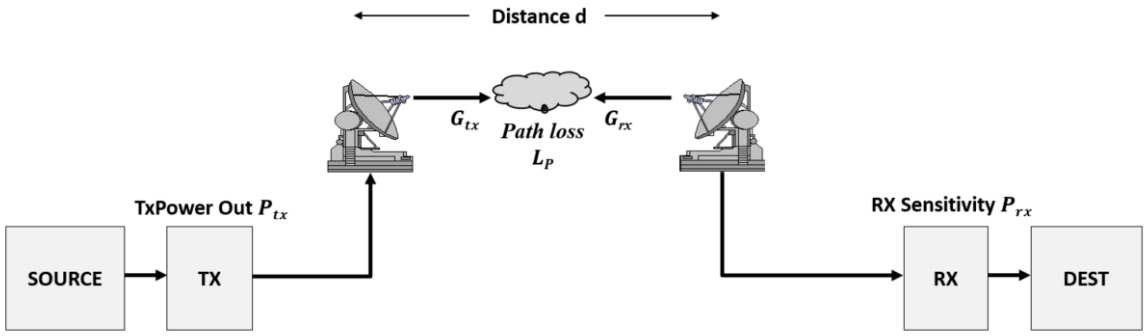


Figure 9: Visualization of (1)

The ideal situation to determine path loss is one where the signal can travel freely through space without ground, tree, building, or other object interference. This is modeled by the Free Space Path Loss (FSPL) equation

$$FSPL = 10 * \log_{10}((\frac{4\pi d}{\lambda})^2) \tag{2}$$

where d is the distance and λ is the speed of light divided by the 2.4 GHz frequency. Using a drone, given a certain height, allows for the optimal scenario as the drone can exist in free skies high enough from the ground and not around trees, buildings,

or other obstacles.

However, free space may not always be a valid assumption or a possible condition. As BLE devices are normally used in households or as human wearable devices and connecting to a device such as a phone at a similar height, ground interference will likely contribute to the path loss. Heydon [15] realizes this condition and provides a formula to estimate the distance BLE communication should travel assuming ground interference, known as the Ground Path Loss (GPL) formula

$$GPL = 40 + 25\log_{10}(d) \tag{3}$$

where d is the distance in meters. Both (2) and (3) are valid equations to use to determine the path loss to input into (1). However, the conditions in the scenario should be considered to choose which is more correct. The link budget equation

$$r = 8.657\sqrt{d/f} \tag{4}$$

where r is the radius for the greatest point in the ellipsis, d is the distance in meters, and f is the frequency of 2.4 GHz, uses the fresnel zone, a cone of radio frequency from one antenna to another [47, 48]. The fresnel zone is an ellipsis from a transmitter to a receiver where a clear Line of Sight (LOS) is necessary to exchange radio signals. While there are many fresnel zones for two communicating antennas, (4) models the radius for the fresnel zone where phase cancellation may be a factor due to obstruction. Additionally, a single fresnel pattern best models, in simplistic forms, the beam of a highly directional antenna, such as the one used in this experiment. If the two devices have an object within that radius, the signal may bounce on the object (such as the ground) to attenuate the signal or to cancel it completely.

2.6 Background Summary

This chapter provides a summary of the BLE technology and the technical specifications for how it works. IoT research and BLE exploitation research are then discussed to show the interest and vulnerability of the field, followed by some vulnerability mitigation tools for BLE. The cyber attack chain is presented, and the chapter is concluded with signal propagation theory.

III. Prototype Design

3.1 Overview

This chapter presents the prototype design for the next iteration of the `skypie` attack platform, `skypie` v3. Section 3.2 provides a summary of the `skypie` v3 overall functions and components. The design goals are then discussed in Section 3.3 followed by hardware in Section 3.4. Section 3.5 covers the software architecture, detailing the design model in Section 3.5.1, `skypie` package in Section 3.5.2, then the `BluBarry` and `skyport` packages in Section 3.5.3 and Section 3.5.4.

3.2 System Summary

The `skypie` is a lightweight, directional Wi-Fi collection and access point exploitation device developed in previous works [1, 2]. The Cyber Network Attack (CNA) payload was equipped with Wi-Fi beacon collection and Global Positioning System (GPS) and geolocation capabilities in `skypie` v1, and was expanded to have Wi-Fi network attack and enumeration capabilities in v2. The previous iterations were designed to be connected atop a drone in order to create an incognito, remote attack platform. This allows the `skypie` operations to function independent of method of locomotion. The sensor can be controlled remotely through a cellular network. The `skypie` v3 sensor prototype is depicted in Figure 10, and now equips the sensor with a long-range Bluetooth adapter and the `BluBarry` package, written in Python 3.7, which allows for BLE device data collection and analysis. Figure 11 shows the hardware components labeled.

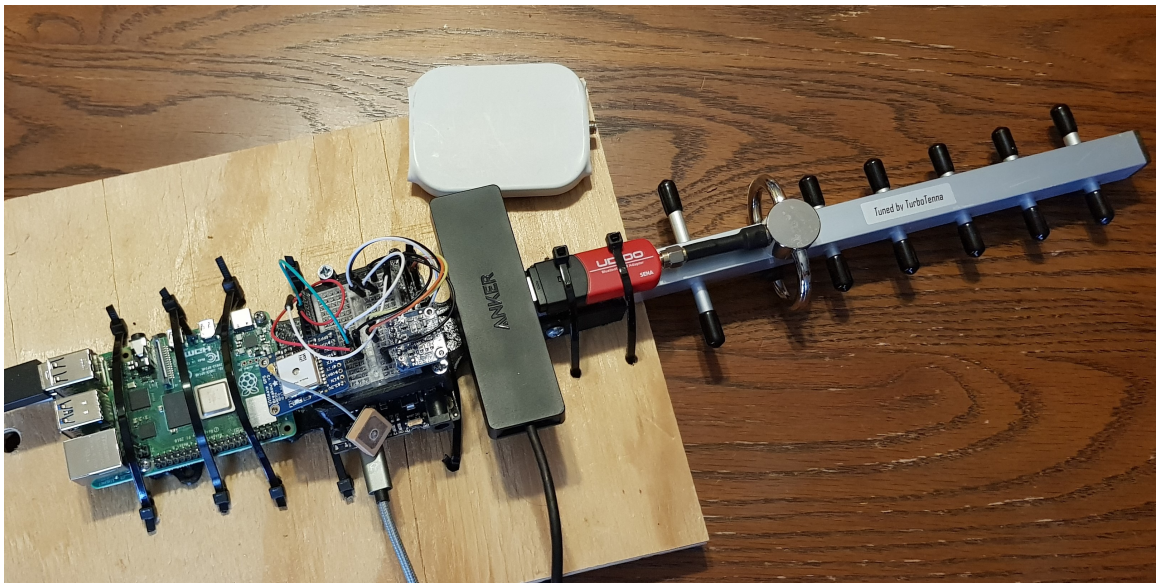


Figure 10: `skypie` v3 Prototype with the Sena UD-100 Bluetooth Adapter

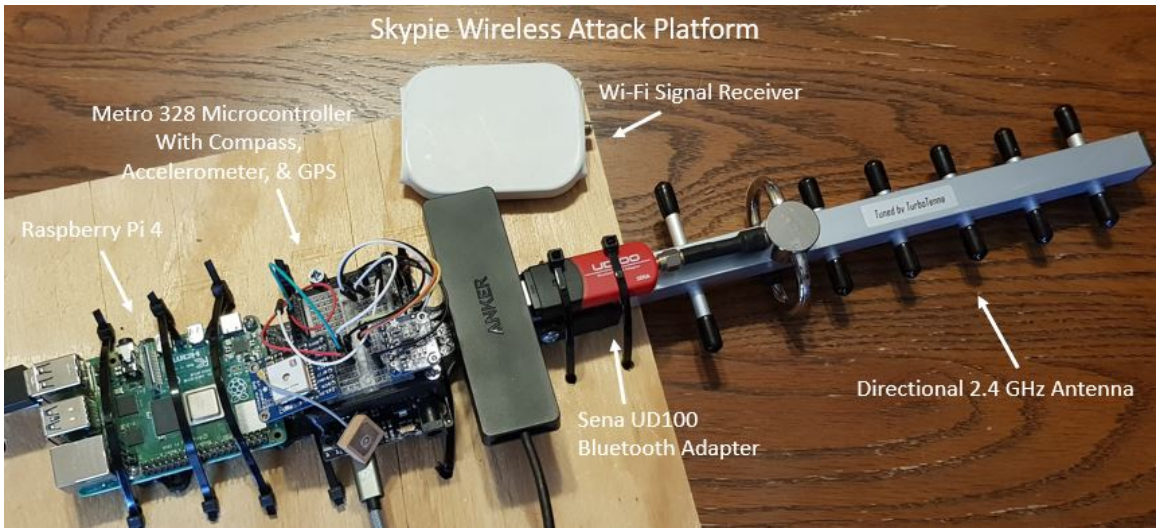


Figure 11: `skypie` v3 Prototype Hardware Components

The `skypie` v3 is a CNA platform with Wi-Fi and BLE collection and attack capabilities that is equipped with its own GPS unit for geolocation. The high-level summary can be seen in Figure 12. The sensor is designed to communicate over a cellular network to drop collection files to a Secure File Transfer Protocol (SFTP) server where the attacker front end, the `skyport`, retrieves the data and displays

it for the attacker. The SFTP server acts as a "dead drop" location so `skypie` and `skyport` do not directly interact with one another. This design is intended to minimize attribution if a component were compromised. `skyport` is a Graphical User Interface (GUI), Python-based web server to host the collected information for the attacker and allow her to send commands to the `skypie`. The sensor can also be controlled using command-line via modifying configuration files.

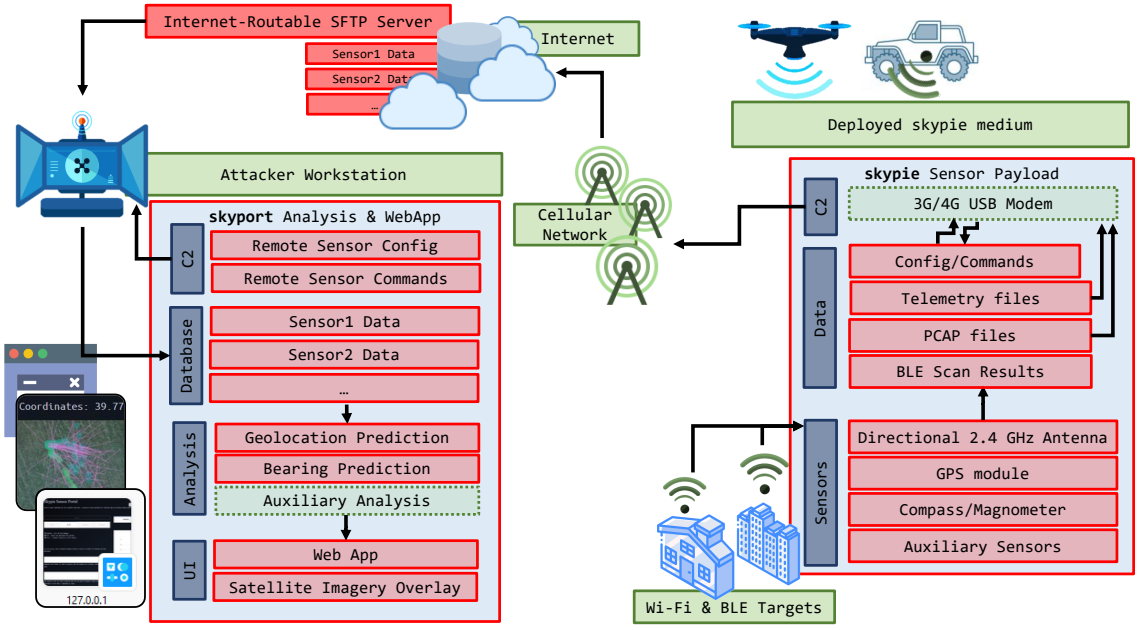


Figure 12: Summary of `skypie` v3 and `skyport` System Design (Adapted from [1])

The `skypie` sensor is designed to capture GPS data (location, speed, angle, etc.) to pinpoint its location. The sensor can then operate in either Wi-Fi mode or utilize the newly developed `BluBarry` package and act in BLE mode. Wi-Fi mode is where `skypie` collects PCAP files with Wi-Fi traffic in it, sends the data back to `skyport` via the SFTP server, then the attacker is presented with the collected information. The attacker can choose a network of interest and begin a series of Wi-Fi attacks where a target device is deauthenticated from the target network and forced to reauthenticate. The malicious actor collects the WPA handshake as the

device reconnects, and through analysis, the attacker can find the key to the network and attach herself to it. Once she is part of the network, she can ping and run an nmap scan on the rest of the network to enumerate it for potential attacks. It was established in Barker's work using `skypie` v2 that these CNAs could be conducted from a distance of 2200 meters away [2].

The BLE mode on this framework scans the surrounding network to provide MAC addresses and device names. An advanced passive scan is then run to determine the RSSIs. The devices which are enumerated in the previous scans are then actively interacted with to determine characteristics and their values based upon the collected MAC addresses. The `skypie` transfers this data to the SFTP server, and the `skyport` downloads it. The attacker can analyze the data and choose to pursue attacks on her newly acquired targets. The command.py module allows the attacker to have full, non-interactive control over the `skypie` and either enact predetermined attacks against a device, or construct a custom attack.

3.3 Design Goals

This research seeks to maintain the integrity of the original design goals [1]. While the attack platform is not being implemented on a drone in this research, the design goals outlined in previous works applies to the focus of this work. The lightweight nature of the `skypie`'s design allows for easy transportation and the ability to remotely deploy the structure on whatever platform (drone, rover, in a car, etc.) necessary. The low-cost constraints allow this work to mirror what a poorly funded but highly motivated actor would be able to develop for attacks, particularly in a contested environment when monetary affluence may be lacking.

- Low Cost: The components that make up the `skypie` prototype should be available to the general market and be inexpensive. Any software not developed

by the attacker must be open source and freely accessible.

- Realistic Utility and Robustness: In order to develop an operational attack platform for realistic CNA, the `skypie` sensor payload must be able to do the following:
  - Operate autonomously or be able to be controlled remotely over a secure wireless channel. If an interruption occurs between the attacker and `skypie`, communication should resume when available.
  - Be capable of near real-time data feedback and control.
  - Collect data and execute relevant attacks for multiple hours. This capability requires adequate storage on the `skypie` sensor and a battery source with sufficient power.
- Portable and Lightweight: As the intent for the `skypie` is that it be movable to the drone or other platform of the attacker's choosing, it must be easily portable. The initial `skypie` v1 design was constructed with a drone architecture in mind; the goal was to allow the `skypie` to be self-sufficient so it could be strapped to any drone that could support the low-weight and not rely on the drone capabilities. This goal is still incorporated in this `skypie` v3 design to allow for the original goal, but also allows the transport of the platform easily to discover new targets. This could be on a drone, on a remote controlled rover, or in a car, and utilized in a plethora of other scenarios.

3.4 `skypie` Hardware Design

The hardware, whose diagram can be seen in Figure 13, is chosen to meet the goals presented in Section 3.3. The architecture remains the same as in previous works, except for the addition of an external long-range Bluetooth adapter and the

upgrade to an 8 GB Raspberry Pi 4 with a 32 GB microSD in order to allow for future capabilities to be integrated. Table 1 lists the hardware used.

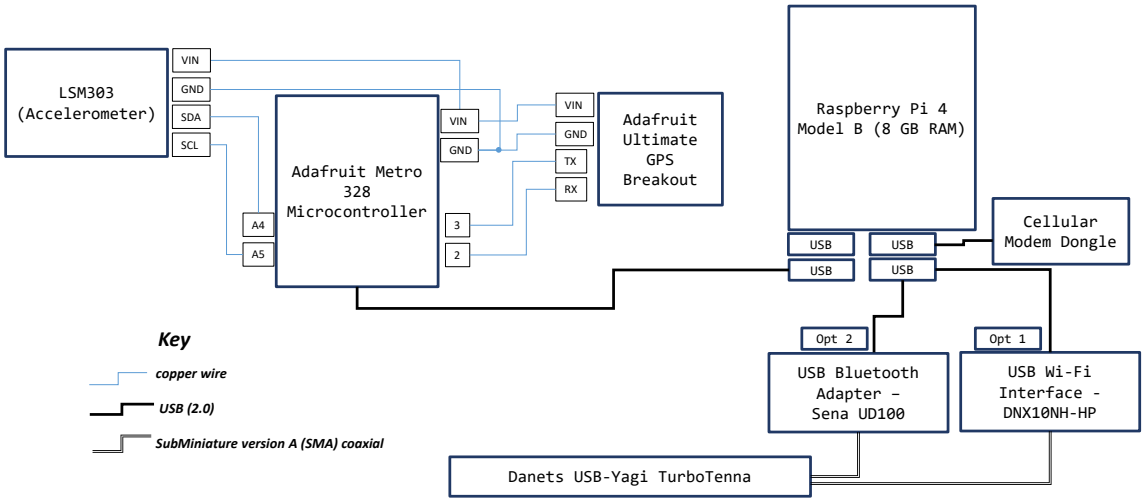


Figure 13: `skypie` Payload Hardware Diagram

The components of the `skypie` payload consist of:

- Directional Antenna: The directional antenna used in this work operates on the 2.4 GHz frequency. Previous works have utilized this device for Wi-Fi collection and attacks. As BLE also operates on this frequency band, the antenna is used to extend the receive and transmit range of a Bluetooth adapter. The gain is approximately 18 dB [49]. This antenna was chosen for its small size, low weight, inexpensive price, and commercial availability.

- Bluetooth Class 1 Receiver: A long-range Bluetooth adapter is being added to the hardware. The Sena UD100 allows for 300 meters with the default stub 1 dBi antenna [50]. This research determines if this adapter can be used at a greater distance for BLE when paired with the directional antenna.

- Wi-Fi Signal Receiver: This receiver came with the Yagi directional antenna, works with the Raspian OS, and is comparable to the Alfa AWUS036H USB

Table 1: `skypie` v3 Prototype Hardware Overview Adapted From Barker's Table [2]

Part	**Model / Version**	**Weight (g)**	**Price**
Directional Antenna	Danets USB-Yagi TurboTenna	137	$110
Bluetooth Class 1 Receiver	USB Bluetooth Interface - Sena UD100	20	$39
Wi-Fi Signal Receiver	USB WiFi Interface - DNX10NH-HP	35	N/A
Computer	Raspberry Pi 4 Model B (8 GB RAM)	46	$75
Digital Storage	32 GB SanDisk Ultra microDSCH UHS-1	1.7	$10
Microcontroller	Adafruit Metro 328	16.5	$18
GPS External Antenna	Passive GPS Antenna uFL - 15mm x 15mm 1 dBi gain	5.5	$4
GPS	Adafruit Ultimate GPS Breakout - 66 channel w/10 Hz updates - V3	8.5	$40
Accelerometer	Adafruit Triple-axis Accelerometer+Magnetometer (Compass) Board - LSM303	2	$15
Power Supply	Charmast 10400 mAh 3 A External Battery Model: W1056	228	$23
4G LTE USB Dongle	Unlocked Modem Huawei E3372-510	18	$30
Optional Sensors	Raspberry Pi Sense HAT	20.4	$38
Structure	3D Printed Casing	52	$2
Structure	Mini-breadboard	13	$2
Miscellaneous Components	Screws, Bolts, Wiring, Headers, USB Cables, Solder	36	$16
Total		**636.6 g**	**$489**

wireless adapter.

- Computer: The computer system is upgraded to the Raspberry Pi 4 with 4 GB additional RAM for a total of 8 GB. This choice does not add significant price increase, but does allow expansion to add more capabilities on the `skypie`.

- Digital Storage: Digital storage for the system has been upgraded to a 32 GB microSD.

- Microcontroller: The Adafruit Metro 328 is chosen due to its ability to do real-time support for hardware modules, which the Raspian operating system (OS) cannot offer. This real-time analysis is required in order to collect accurate GPS data.

- GPS: The Adafruit Ultimate GPS Breakout v3 was chosen to provide real-time GPS data to the system. It has a signal sensitivity up to -165 dBm, 66 channels, and provides jammer detection and reduction. It is equipped with a 1 dBi external antenna [51].

- Accelerometer: The Adafruit Triple-axes Accelerometer+Magnetometer (Compass) Board uses the LSM303 chip and was chosen for its small size, user-friendly interactions, pre-built libraries, and calibration settings.

- Power Supply: The Charmast W1056 battery was chosen to provide 5V/3A to the Raspberry Pi 4.

- 4G LTE USB Dongle: The Unlocked Modem Huawei E3372-510 was chosen in previous work due to its compatibility with the Raspian OS.

- Optical Sensors: The optical sensors with the Pi Sense Hardware Attached on Top (HAT) allow the attacker to visually know what state the `skypie` is in during its operations without requiring a screen.

- Structure: The encasing on the `skypie` is custom 3D printed to allow for the microcontroller with GPS unit, the Raspberry Pi, and the top of the antenna to be secured together, and for the directional antenna with the Bluetooth adapter to have unobstructed access. The structure can be seen in Figure 14.

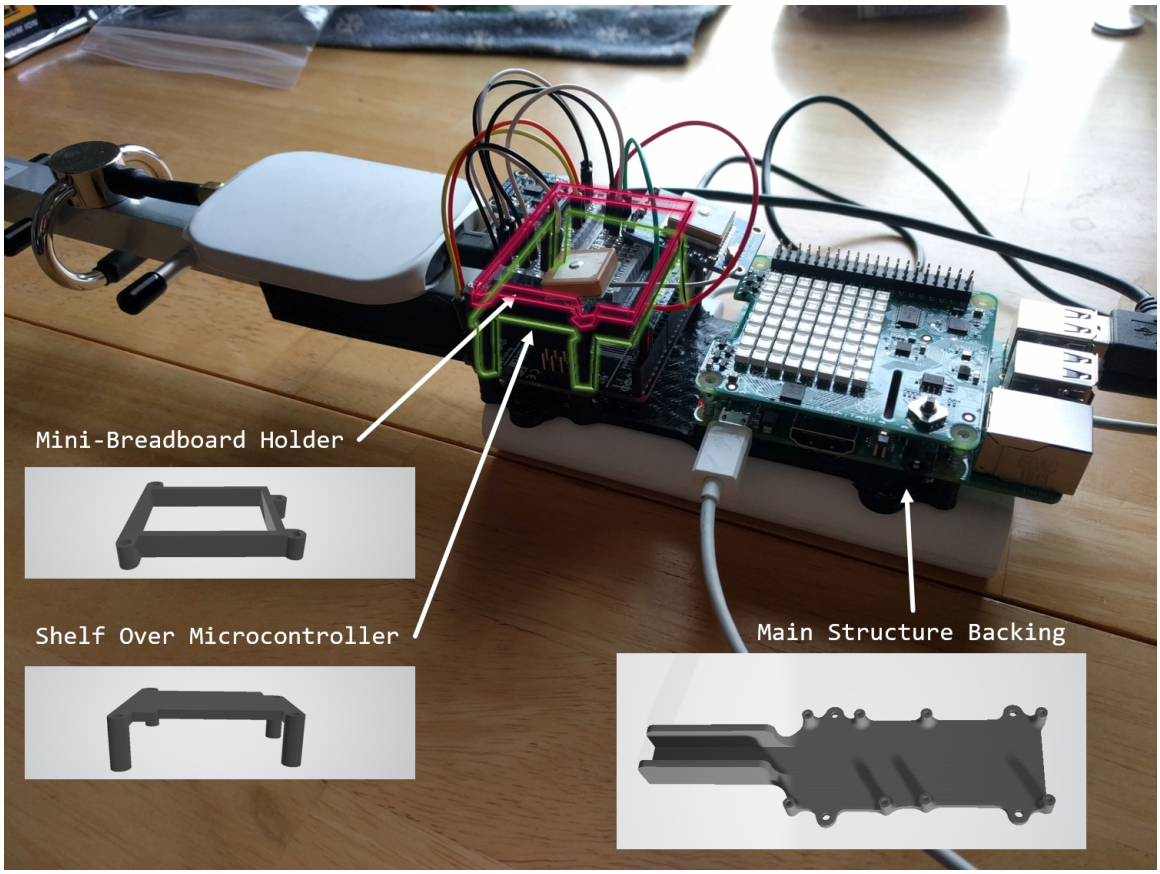


Figure 14: 3D Printed `skypie` Structure [1]

3.5 `skypie` Software

The `skypie` prototype software is written in Python 3.7, with the exception of the microntroller code and geodesic intersection algorithm, which are written in C++. The C++ code is not altered in this research and remains in the repository. The system repository is divided into three packages: the `skypie`, the `skyport`, and shared, which are modules used by both `skypie` and `skyport`.

3.5.1 Design Model

The design model for the `skypie` software in its initial iteration and v2 are upheld while developing `BluBarry` for `skypie` v3 and integrating it into the existing

system.

- The `skypie` is controlled via a configuration file that includes the necessary instructions to tell the sensor what data to collect, any additional commands to execute, etc. The default config file, along with options, can be found in Appendix A.

- The payload operates through a control loop, where the most current version of the config file is checked and machine state is altered during each iteration.

- The latest config file includes specifications for what state alterations are to be run. Subtask examples are what Wireshark filters to use, MAC addresses for Wi-Fi or BLE targets, timeouts, nmap search parameters, and intervals.

  - Any changes to the config file are only implemented once the current threads have expired to bolster the stability of the system in accordance with Python programming best practices.

  - The attacker has the ability to modify this file to adapt to new environments or focus on new targets using the `skyport` web GUI.

- Each task acts as its own asynchronous thread in order to allow multiple operations to be completed individually.

- The `skypie` collects the requested data, such as for Wi-Fi or BLE, in its own file, which is named with the thread start time, and whose size is dictated by the collection interval or number of iterations chosen.

- In order to lessen the opportunity for data corruption, and to maximize code readability and a simpler implementation with better coding principles, the timespan of the threads is specified. While this may result in lost data be-

tween the termination of one thread and the start of a new thread, the benefits presented outweigh the potential loss.

- The SFTP server acts as "dead drop" point in order to minimize attribution between the `skypie` and `skyport`, and as both end points are unlikely to have static Internet-facing IP addresses.

- The analysis of GPS positioning, and Wi-Fi and BLE data, as well as Wi-Fi network cracking, is performed on attacker's workstation. `skyport` is managed through its own control loop responsible for retrieving files, sending updates to the config file for commands meant for the skypie, sorting data from the files, and performing Wi-Fi cracking.

- To maintain stability with the `skypie` payload, a cron job is designed to reinitiate after 60 seconds of inactivity. If the attacker wants to stop the sensor from collecting, she could choose the "off" mode option in the `skyport`, in which case the sensor would still check every 60 seconds for an update, and if still in "off" mode, sleeps until the next 60 second increment.

- The `skyport` user interface should be user-friendly in order to show capacity for operational use for cyber attacks.

3.5.2 `skypie` package

Table 2 shows the required dependencies and descriptions for the new iteration of `skypie`–`skypie` v3.

Figure 15 demonstrates the `skypie` code design and the restructuring due to `skypie` v3 upgrades. The primary files integral to the system and to the `BluBarry` package integration are described in the following sections. The modules not described

Table 2: skypie v3 Dependencies

Package	Function
aireplay-ng	Injecting generated packets (i.e., deauthentication packets)
airodump-ng	Targeted WPA handshake capture
btmgmt	Python wrapper to interact with BlueZ stack, used to detect BLE devices and determine RSSIs
dumpcap	Capture packets from Wi-Fi interface
gatttool	Attempt to connect to BLE device, query for current settings, request to write
hciconfig	Get Bluetooth adapter settings, set interface of interest
hcitool	Use the HCI layer of the adapter to scan for BLE devices
iwconfig	Get Wi-Fi adapter settings, set monitor mode/channel
ifconfig	Prepare wireless network interface controller for monitor mode
iwlist	Get Wi-Fi interface current channel
nmap	Conduct network scans
tshark	Makes packet captures available to Python for analysis

in this thesis can be read in detail in [1] and [2]. Other threads have been modified to allow for the enhanced capabilities presented with skypie v3, but are not discussed.

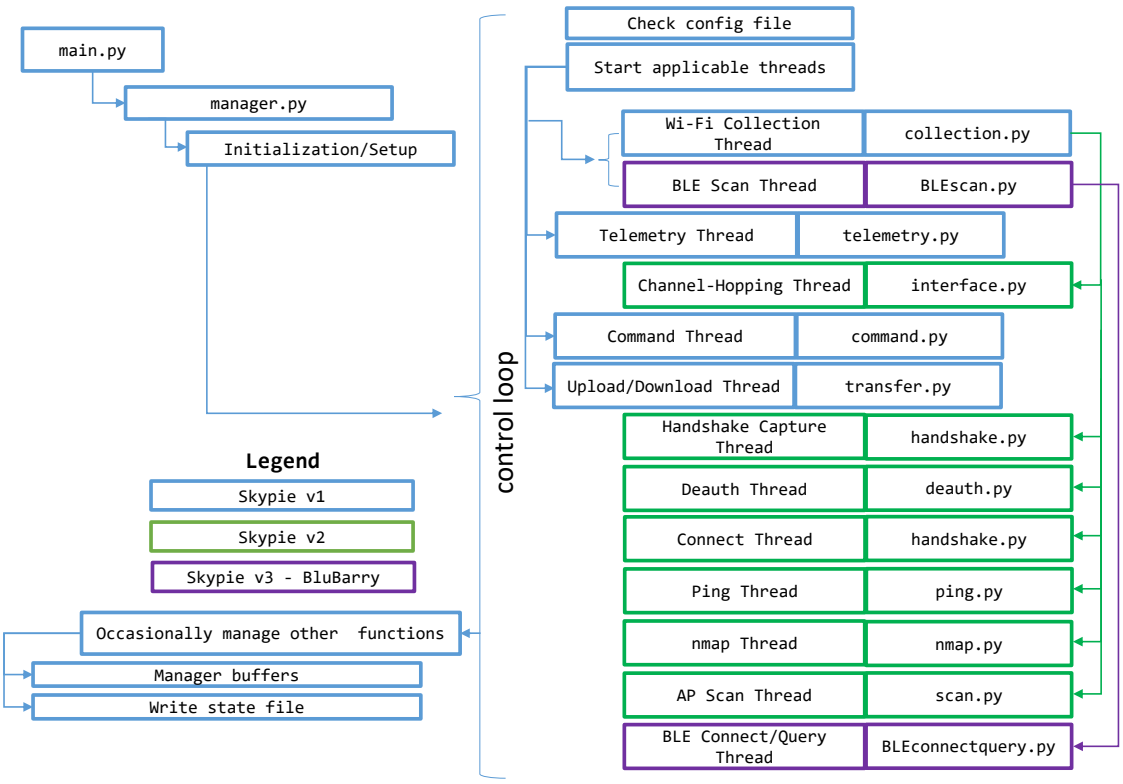


Figure 15: Software Control Flow Diagram

3.5.2.1 main.py

Main is the startup program that begins upon initialization of the `skypie`, or manually by the attacker. The file constructs the requirements for manager.py based upon the startup parameters. The main control loop can be run with the following options:

- `-a` to activate the main loop, requiring GPS
- `-n` can be run in conjunction with `-a` to negate the GPS satellite fix requirement
- `-d` shows debugging options and can be run with the previous two commands
- `-x` initiates self-destruct mode; in the case that the `skypie` is compromised, the attacker can send a self-destruct mode to destroy the data on the pi

3.5.2.2 manager.py

This module runs the control loop for the program. The manager specifies in what order the threads are run. It acts according to the parameters set forth in the default configuration file. Manager creates a dictionary of threads for each independent function. The manager checks the config file at the beginning of each loop. If the attacker updates the file, the new parameters are applied for the next iteration. Each thread is run for a specified amount of time, and once that thread has completed, the manager spawns a new thread unless the config file stipulates otherwise. The option chosen by the attacker to focus on Wi-Fi or BLE is set in the config file, and the manager initializes or disables the corresponding adapters.

3.5.3 `BluBarry` package

The `BluBarry` package lives within the `skypie` package and is the option to allow the attacker to focus her attention on Bluetooth Low Energy.

3.5.3.1 BLEscan.py

This module, a copy of which can be found in Appendix B, is responsible for scanning and collecting BLE advertisement data. This is done passively through received data only, and does not require interaction with the targets on the part of the `skypie`. The Sena UD100 Bluetooth adapter, connected to the directional antenna, allows for long-distance, passive collection of this data. The adapter is programmed natively to hop the three advertisement channels discussed in Section 2.2.1.1, channels 37, 38, and 39. The adapter then collects the advertisement packets and parses them for the requested data. The command

```
hcitool lescan
```

is run in order to do a light scan to gather the advertisement packets of detected devices. This command outputs the MAC and name of the device, which is displayed on `skyport` for the attacker to see what devices are nearby. If an output error is detected, the commands

```
hciconfig hci0 down
hciconfig hci0 up
```

are run to take down the Bluetooth interface and bring it back up. A more intense (but still passive) scan is then run using the command

```
btmgmt find
```

Among the advertised information is the MAC, whether the device is LE or BTC, RSSI, flags set, and device name. The RSSI value can be used for approximate distance estimation and to prioritize which devices may be closer than others. The attacker has the option to choose how many iterations this `btmgmt` command is run. The RSSIs are averaged over the collected values, giving the attacker both the raw data collected and the overall average. The number of times a device advertises itself is dependent on manufacturer settings and how recently the device has been turned on or used, so this research has no control over how many beacons are collected. However, iterating the commands more times increases the likelihood that more beacons are detected, allowing for more RSSIs to be averaged. The results of the command are parsed and organized into a database object. Once a particular MAC is detected, the MAC, type, flags, and device name are stored in the database. The RSSIs are collected each iteration and appended to that MAC along with the time in Universal Time Code (UTC) that a particular RSSI was collected. This database is unpackaged for attacker view on `skyport`.

3.5.3.2 BLEconnectquery.py

Once blescan.py collects device data advertised passively, the `skypie` transmits back to devices collected in the previous scan. The database assembled in blescan.py is passed into this module, and the `gatttool` command line tool is used to attempt to connect and query devices from the database for device characteristics. Whether or not a device is connectable in this manner is dependent on how the manufacturer has configured the BLE device. If connection is possible, the command

```
gatttool -b [MAC] --characteristics
```

is run to list all handles, char properties, char value handles, and uuids. This information is stored in the database, then each characteristic is read by handle with the command

```
gatttool -b [MAC] --char-read -a [handle]
```

These handles hold descriptors that contain information such as device name, manufacturer, serial number, current battery levels, brightness, sound level, etc. For example, the MiPow Playbulb handle `0x0016` holds the device name, and `0x0010` holds the current brightness of the lightbulb. All characteristics are read on the device as it is not predictable what handle will hold what information. This is determined by the manufacturer during setup and is device dependent. This data is stored in the database and sent to the SFTP server to be gathered by the `skyport` for attacker analysis. The code for BLEconnectquery.py is listed in Appendix C.

3.5.4 `skyport` Package

The `skyport` interface allows an attacker to specify the type of information she would like to collect with the `skypie`. She must choose the BLE option in order to collect the relevant data and analyze it for further actions.

3.5.4.1 sensor_app.py

The sensor_app module dictates the attacker interface design. The original version developed in [1] included settings to control the `skypie` (known as the sensor), a Wi-Fi tab for overall Wi-Fi collection, and a telemetry collection tab. The next iteration in [2] added a Wi-Fi attack tab in the control settings. `skypie` v3 comes with an upgrade to the `skyport` where a BLE tab is created to update the configuration file used by the `skypie`. The settings tab options can be seen in Figure 16. Additionally, the attacker can send a predefined attack, further explained in Section 3.5.4.2.

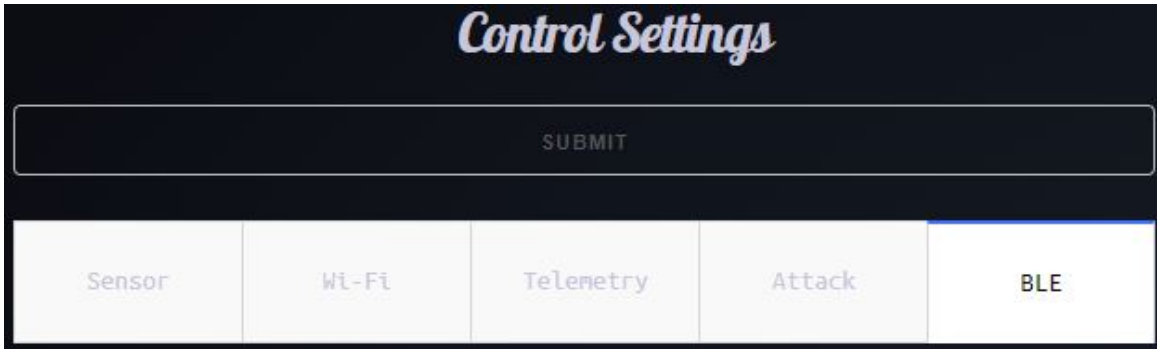


Figure 16: `skyport` Control Settings Tabs

The data collected by the `skypie` displays on the `skyport` under the specific sensor name (in this research, the `starchy` sensor is used). The tab options are displayed in Figure 17. The telemetry and calibrate tabs are used for the `skypie`'s geolocation capabilities. The targets tab displays the Wi-Fi networks collected by the sensor in a table for attacker view. The console tab allows an attacker to send commands to be run on the `skypie` and are discussed in Section 3.5.4.2. The log tab presents the output log as the `skypie` is running.

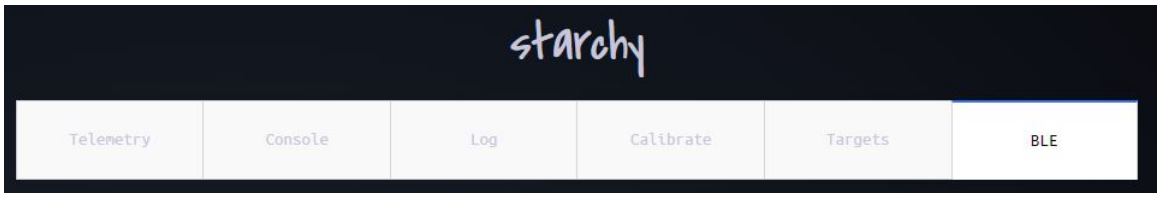


Figure 17: `skyport` Tabs to Display Data and Send Commands to Console

The `skypie` uploads the database collected by the `BluBarry` modules to the SFTP server, where the `skyport` downloads it and displays it in the BLE tab. Figure 18 shows that the attacker can see the name of the device, the MAC, the averaged RSSIs, as well as the characteristics for the device if it is connectable. The attacker can choose what device and characteristic to pursue and attempt to overwrite the characteristics at the user's choosing. Depending on the security of the device programmed by its manufacturer, such attacks may or may not be successful.

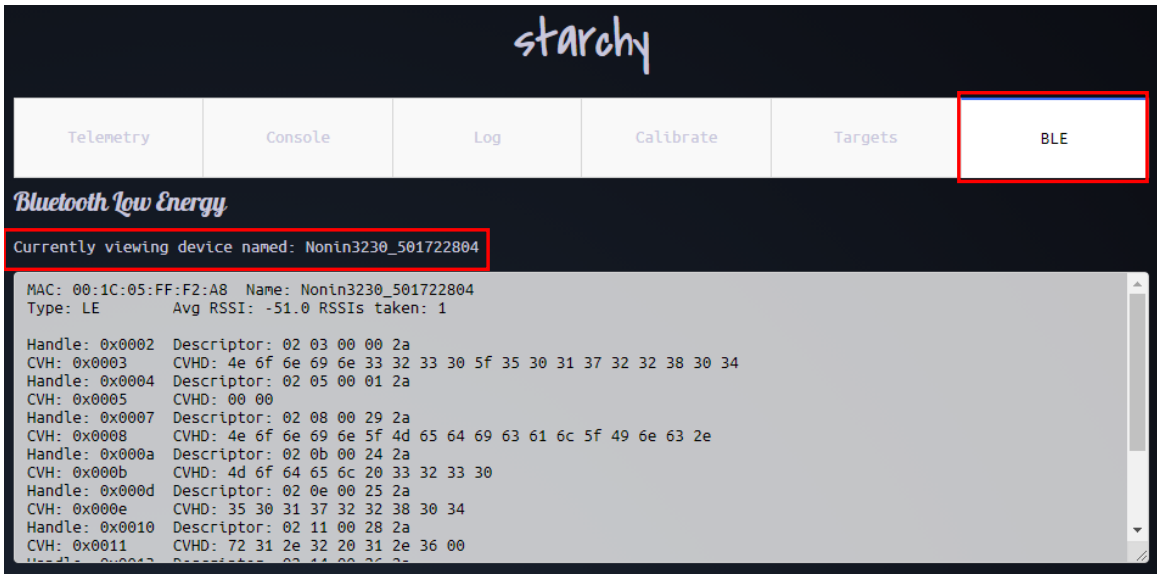


Figure 18: `skyport` BLE Tab to Display Database Results

3.5.4.2 command.txt

The `skyport` offers an interface for the attacker to enter commands to be executed on the `skypie`. Figure 19 shows the output of an `ifconfig` command.

Once a command is entered on `skyport`, it is put into the command.txt file and uploaded to the SFTP server. The `skypie` pulls down the command file at the interval specified by the attacker in the config file. Once downloaded on the `skypie`, the commands are run and the output put into the consoleLog.txt and sent back to the SFTP server. The `skyport` downloads the console file, and the results are displayed for the attacker in the console tab. This particular feature allows the attacker to run any command desired on the `skypie`.

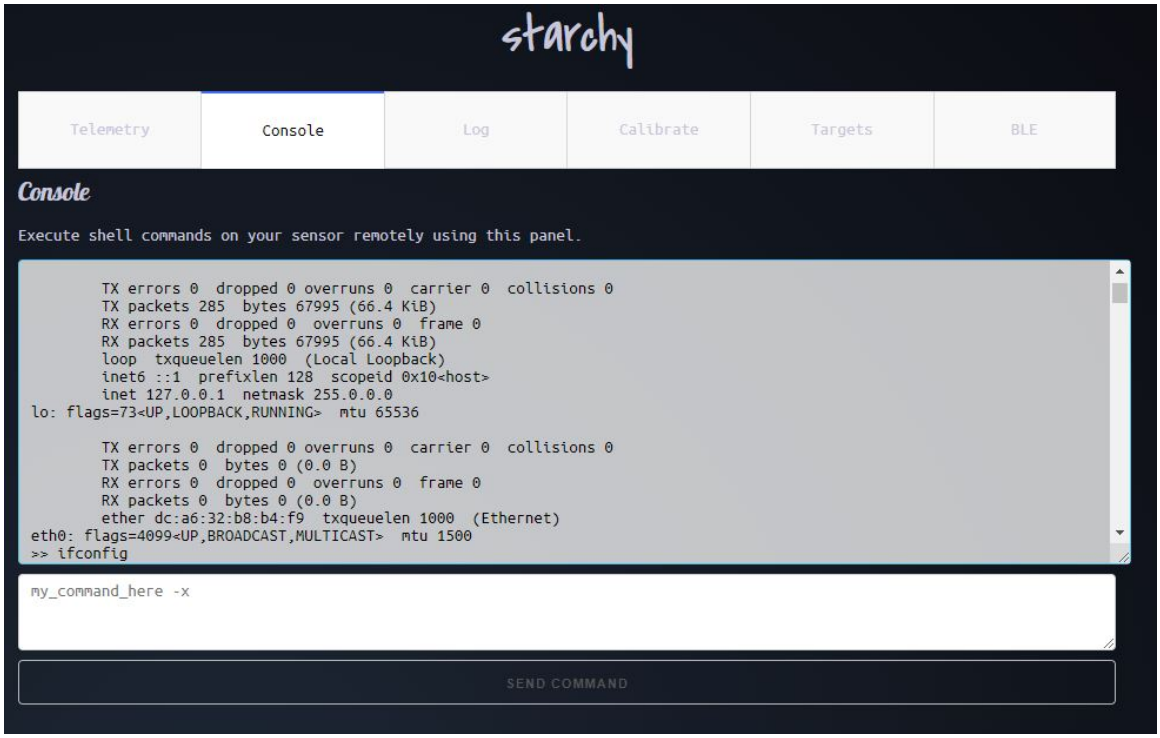


Figure 19: `skyport` Interface for Console Commands to `skypie`

The BLE control settings tab, shown in Figure 20 allows the attacker to craft a specific attack to attempt to overwrite characteristics on a target device with the command

```
gatttool -b [MAC] --char-write-req -a [handle] -n
[newcharacteristicvalue]
```

Figure 20: `skyport` Interface for BLE Write Attack Parameters to `skypie`

The command is executed once the command file arrives on the `skypie`. The response is sent back to the attacker to notify if the characteristic value was written successfully or if there was a failure. Errors may be experienced if the attacker is

attempting command values that are not expected for the characteristics or that are not the correct length. If successful, the attacker can view the new value in the BLE tab as the `BluBarry` package is continually running the scans and queries unless told otherwise. The attacker can also choose the command

```
gatttool -b [MAC] --char-read -a [handle]
```

from the BLE interface option to read a specific value associated with the handle of interest. Changing the values can have a physical effect, such as in the example of the Playbulb mentioned in Section 3.5.3.2, an example of which is shown in the `skyport` in Figure 21. In this case, the brightness of the lightbulb was at `13 13`, which means it was almost at its full brightness. The attacker was able to send the value `0000` to this characteristic, which dimmed the lightbulb to `12 12`, just slightly darker than previously. This effect of changing the brightness could be visually confirmed while watching the lightbulb. If the value changed is the name of the BLE device, the user may be required to manually re-pair with that device, so it could result in a temporary DoS to the IoT device. The attacker can utilize this time to change more characteristics on the device to wreak more havoc before the user has time to react.

Figure 21: `skyport` Console Tab for Attack Output

3.6 Design Summary

This chapter has discussed the `skypie` v3 prototype components and design. Section 3.2 summarized the SUT, followed by design goals and hardware design in Section 3.3 and Section 3.4. The `skypie`'s software is discussed in Section 3.5 which dives into the design model for the software, then the `skypie`, `BluBarry`, and `skyport` packages.

IV. Methodology

4.1 Overview and Objectives

This research extends the work previously done by Bramlette [1] and Barker [2] to integrate BLE attack capabilities into the `skypie` attack platform, and by focusing on the following questions:

- How close does an attacker need to be to collect Bluetooth Low Energy (BLE) data from a target using lightweight, low-cost equipment?
- Can pattern-of-life data be collected at 600 meters?
- At what distance does pattern-of-life collection become infeasible?
- Can the attacker also interact with the target from that distance?
- How is the range of the `skypie` affected at a height of 1 meter versus 3.05 meters?

The System Under Test is explained in Section 4.2, followed by the experiment factors in Section 4.3, metrics in Section 4.4, constant parameters in Section 4.5, and uncontrolled variables in Section 4.6. Experimental design in Section 4.7 concludes the chapter.

4.2 System Under Test

The System Under Test (SUT), as described in Chapter III, is designed to update an attack platform, the `skypie`, that is made with low-monetary investment and with lightweight equipment. Additionally, the platform allows for remote control so an attacker does not necessarily need to be at the same location as the `skypie` and can

control it from an undisclosed location. The platform has incorporated geolocation via radiolocation up to 600 meters [1] and Wi-Fi collection and exploitation up to 2200 meters [2]. Figure 22 summarizes the overall SUT.

4.3 Factors

This experiment focuses on the factors listed in Table 3. These are the parameters that are varied throughout the experiment.

- Distance: The distance, measured in meters, is evaluated from 50-600 meters at 50 meter increments. This allows for a total of 12 measurements throughout the experiment. However, if no beacons are collected from any of the devices, that data point is dropped.

- Antenna Mode: The antenna functions first as receive (RX) only. This is modeled through passive beacon collection to determine if the attack platform can sniff data from a certain distance. RX and transmit (TX) is the other mode, used to interact with the target device and attempt connection and characteristic enumeration.

- Elevation: The `skypie` is evaluated at elevations of 1 and 3.05 meters. The 1 meter height is to determine the effectiveness of the platform for potential uses other than drone flight. The 3.05 meter height simulates the height of a drone flying to test the `skypie` v3 effectiveness by the means evaluated in the previous iterations.

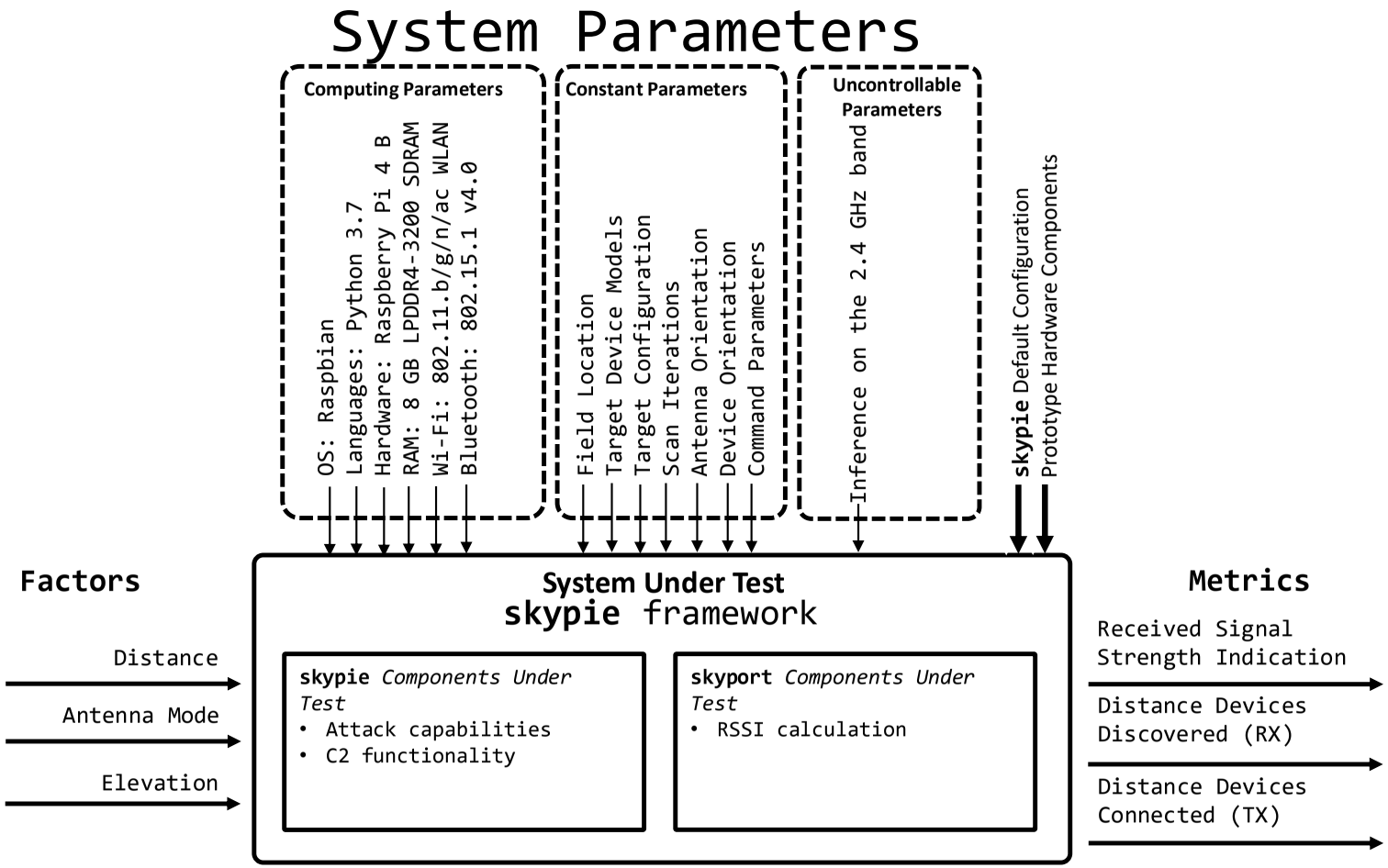


Figure 22: System Under Test Diagram

Table 3: Experiment Factors

Factor	Levels	Description
Distance	50-600 meters (50 meter increments)	The distance from the attack platform to the target devices
Antenna Mode	Receive (RX) only, RX & Transmit (TX)	The manner in which the antenna is utilized
Elevation	1 meter 3.05 meters	The effectiveness of `BluBarry` at different heights

4.4 Metrics

The metrics to determine experiment success, described in Table 4, are Received Signal Strength Indicator (RSSI), distance devices discovered for RX only, and distance devices connected for RX and TX.

- Received Signal Strength Indicator (RSSI): useful in measuring the approximate distance between the two devices (`skypie` and the target device). The value is determined based upon the power level received by the `skypie` antenna. RSSI is unitless, but can be converted to dBm representing milliwatts (mW) [46]. The logarithmic relationship between dBm and mW is

$$mW = 10^{\frac{dBm}{10}} \tag{5}$$

  The Bluetooth SIG recommends using multiple RSSI's averaged at a distance to determine the true RSSI value, as a single RSSI reading could be influenced by interference. Section 2.5 details how RSSI can be estimated to determine accuracy against theorized RSSI and actual RSSI.

- Distance Devices Discovered: the distance at which the target devices beacons can be collected passively, utilizing the antenna mode to receive only. A col-

lected BLE beacon includes the name of the device, the MAC, and the RSSI. This information is used to determine pattern-of-life data, and thus beacons and pattern-of-life collection are used interchangeably.

- Distance Devices Connect: the distance that a target device can be transmitted to and receive data from. If the `skypie` is able to detect the target at the distance, it will attempt to interact with it through a connection.

Table 4: Experiment Metrics

Metric	**Units**	**Expected Range**
Received Signal Strength Indicator (RSSI)	dBm	$-82\ dBm \leq RSSI \leq -50\ dBm$
Distance Devices Discovered	meters	$50 - 600$
Distance Devices Connected	meters	$50 - 600$

4.5 Constant Parameters

The following are parameters that remain constant as the experiments are conducted. This is done in order to reliably determine the success of the experiment and not to wrongly attribute any effects due to environment or equipment changes that are not measured. The constant parameters are summarized in Table 5.

- Location: The location chosen, seen in Figure 23, allows for a 800 meter range on a flat, open area. The field is located between the Air Force Museum and Loop Road. This location is optimal in order to minimize 2.4 GHz range interference in the immediate area and allow for continuity of experiment runs. This location was also chosen for proof-of-concept for `skypie` v1 experiments [1].

Table 5: Constant Parameters

Parameters	**Proposed Values**	**Controlled By**
Location	Open Field (800 meters)	Experiment Design
Number of Targets	3	Experiment Design
Target Devices	Masimo MightySat Rx, Nonin 3230, & SensoSCAN S340	Experiment Design
Device Orientation	x orientation	Experiment Design
Antenna Orientation	Direct LOS to targets	Experiment Design
BLE Scan Iterations	30 iterations per collection	Device Configuration
BLE Scan Command Parameters	`hcitool lescan` `btmgmt find`	Device Configuration
Connection & Enumeration Command	`gatttool -b [MAC] --characteristics`	Device Configuration
Characteristic Probe	`gatttool -b [MAC] --char-read -a [handle]`	Device Configuration

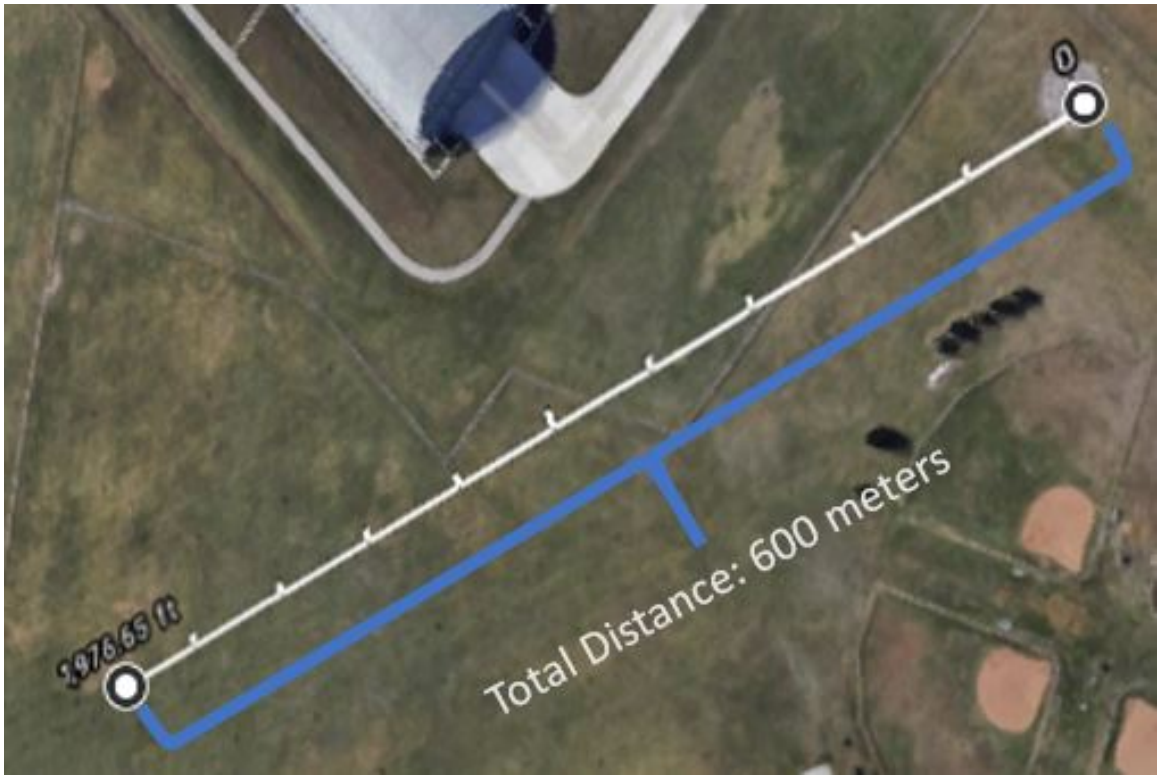


Figure 23: Experiment Location

- Number of Targets: Three targets are chosen for the two experiments to determine the effectiveness of the `skypie` against three similar types of devices.

- Target Devices: The target devices for these are the medical IoT devices - the Masimo MightySat Rx, Nonin 3230, and SensoSCAN S340 seen in Figure 24 with quarters for reference to show size. The target devices are fingertip pulse oximeters using BLE versions 4.0-4.2. For the purpose of making this research applicable to battlefield research, the Masimo device was chosen as one of the targets for this research. This device is one of the devices utilized by the BATDOK project according to the official website [24].

Figure 24: Target Medical IoT Devices

- Device Orientation: Per pilot studies discussed in Section 5.2, the devices are fixed along the x axis during all data collection.

- Antenna Orientation: The Yagi antenna on the `skypie` is pointed with LOS to the target devices and remains in that position throughout the experiments.

- BLE Scan Iterations: Each collection point in both experiments are run at 30 iterations. This can vary in time based upon the number of devices detected, but was around 7 minutes for this research. As the `BluBarry` functionality of the `skypie` interacts with the device without the user's knowledge, detection is not a large concern. However, if an attacker were to remain in one location for too long, it increases the chances of being identified. Therefore, 30 iterations was determined to be an acceptable amount of time. The attacker has the ability to change this and do a shorter or longer scan.

- BLE Scan Command Parameters: The passive light and intense scan parameters remain the same throughout the experiments as discussed in Section 3.5.3.1.

- Connection & Enumeration Command: The `gatttool` command is used to attempt to connect to and list out the characteristics on the device.

- Characteristic Probe: The characteristic probe command is repeated for each data collection and attempts to read the handle of each characteristic found by the previous command, as detailed in Section 3.5.3.2.

4.6 Uncontrolled Variables

The 2.4 GHz frequency band is used by numerous wireless protocols such as Wi-Fi, Bluetooth, Bluetooth Low Energy, Zigbee, garage doors, radio toys, and more. This means that the open ISM band is subject to interference. Though the location to conduct the experiments was chosen to minimize this, any traffic on this band simulates an operational environment.

4.7 Experiment Design

The following sections describe the two experiments conducted in this thesis. The experiments address the following research questions:

- How close does an attacker need to be to collect Bluetooth Low Energy (BLE) data from a target using lightweight, low-cost equipment?
- Can pattern-of-life data be collected at 600 meters?
- At what distance does pattern-of-life collection become infeasible?
- Can the attacker also interact with the target from that distance?
- How is the range of the `skypie` affected at a height of 1 meter versus 3.05 meters?

4.7.1 Experiment One: `skypie` Elevation 1 Meter

1. The same location is utilized for all experiments.
2. Target devices are placed at the 50 meter distance at a height of about 1 meter. The devices are at the same elevation and same orientation during data collection.
3. The attack platform is placed on a plastic table at an elevation of 1 meter, shown in Figure 25. The antenna is placed with a direct LOS to the target IoT medical devices.

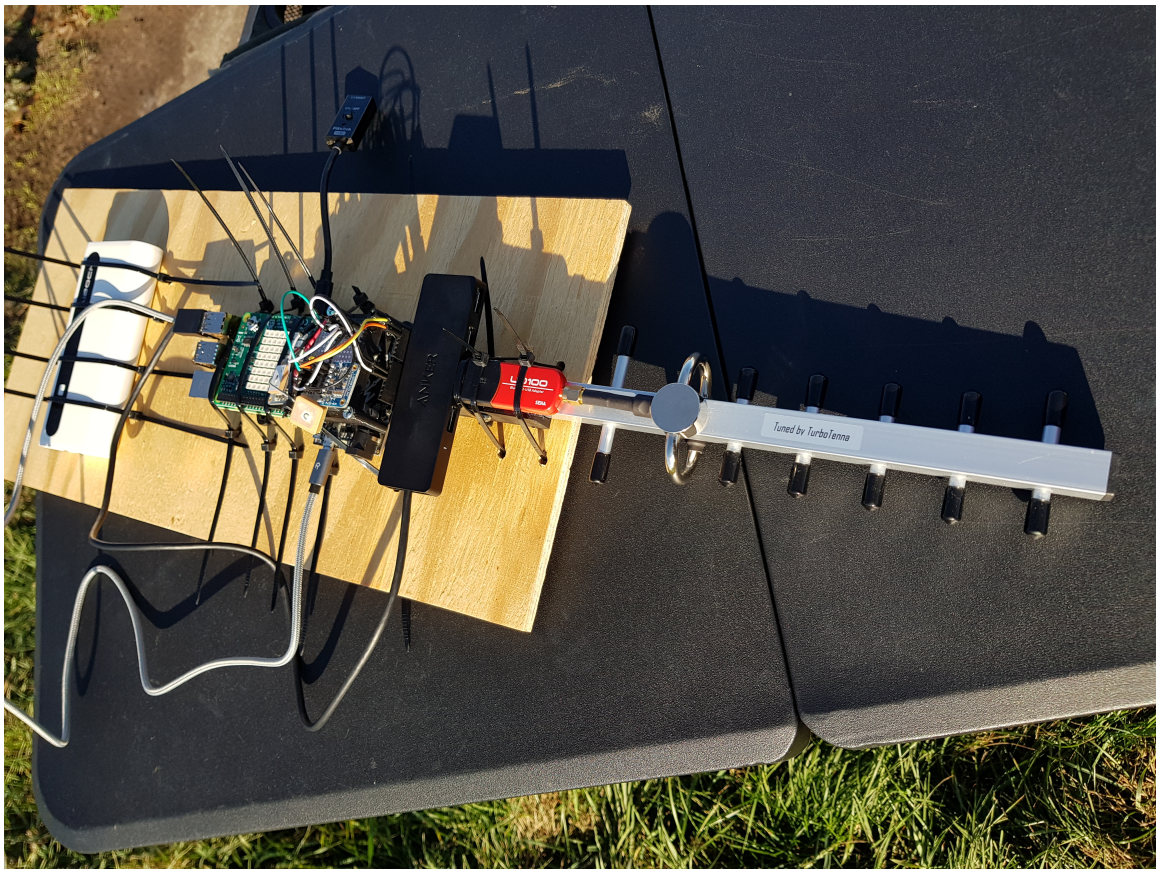


Figure 25: Attack Platform at 1 Meter Elevation.

4. The attacker logs in to the `skyport` to begin interacting with the attack platform, when necessary. If not already configured, the attacker chooses the BLE Attack option in the `skyport`.

5. The `skypie` prototype is turned on, with the long-range Bluetooth adapter in position. The `skypie` software begins automatically and runs according to the latest version of the config file. Any updates to the config file requires a cellular connection for the `skypie` to receive commands from the `skyport` and also to send data back to the SFTP server. However, as the experiment is run in proximity to the `skypie`, a hotspot is used to interact with the `skypie` from the `skyport`. The software is initialized as described in Chapter 4, and the

BLE data collection and attack begins.

6. The `BluBarry` package on the prototype is set to iterate 30 times through the intense scan to populate the database for nearby BLE devices.

7. The `skypie` then begins the BLEconnectquery thread and attempts to connect to the target devices and enumerate current characteristics settings.

8. Target devices are moved to the next 50 meter checkpoint and the experiment is repeated until the 600 meter distance is met and evaluated.

4.7.2 Experiment 2: `skypie` Elevation 3.05 Meters

Experiment two is run in the same sequence of events as experiment one. The `skypie` is now mounted on a pole and elevated to 3.05 meters to simulate realistic drone flight, as seen in Figure 26. The pole compared to a 1.22 Meter Ruler can be seen to show the relative height. The prototype is vectored at the target devices with a direct LOS and its orientation not changed throughout the experiment.

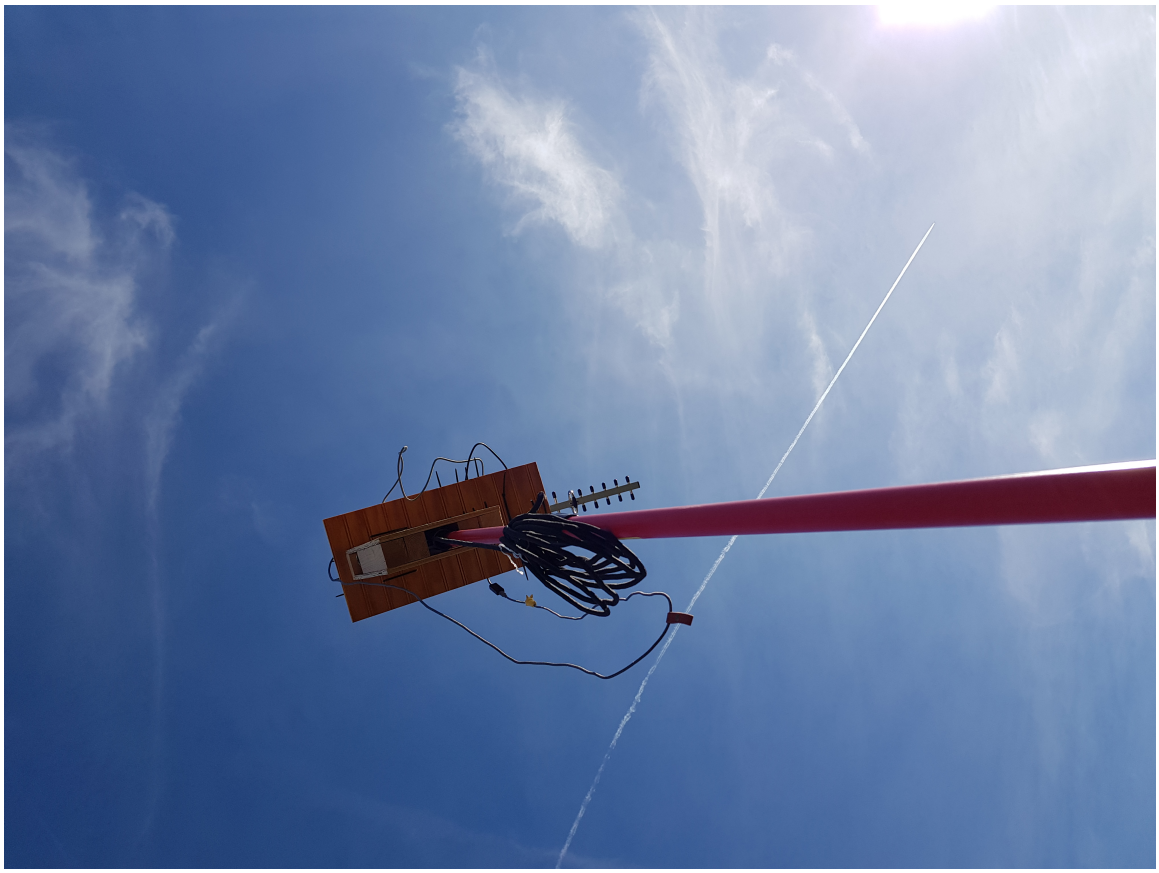


Figure 26: Attack Platform on Pole at 3.05 Meters for Experiment Two.

Figure 27: Pole Compared to a 1.22 Meter Ruler.

4.8 Summary

This chapter describes the System Under Test (SUT) used including factors, metrics, and parameters for the experiments, as well as the steps for how the experiments

are conducted. The experiments depicted here are done in order to determine the distance of BLE attacks on a lightweight, low-cost attack platform. The data collected is analyzed to determine statistical significance and to judge the effectiveness of the experiments against the metrics in Chapter 5.

V. Results and Analysis

5.1 Overview

This chapter discusses the results obtained from the experiment outlined in Chapter 4. Section 5.2 discusses the orientation of the devices for the experiments. Section 5.3 details the results of the beacon collection and connection attempts and discusses the overall RSSI trends. Section 5.4 concludes the chapter by calculating the expected RSSIs using (2) and (3) for path loss analysis. Section 5.4 also includes a statistical analysis of the Nonin and S340 device RSSIs in each experiment to determine if height had a significant effect on the RSSIs collected.

As was determined by previous `skypie` experiments [1, 2], the `skypie` attack framework was able to geolocate at 600 meters and perform Wi-Fi network focused attacks up to 2200 meters. This research sought to expand the capabilities of the `skypie` attack platform and `skyport` attacker controller interface by integrating the `BluBarry` package, allowing BLE device detection, enumeration, and modification. The results of the experiment are directly responsible for answering the research questions in Section 4.1 to determine the capabilities of this package.

The results discussed in this chapter compare two experiments: one run at a height of about 1 meter and the other repeated at 3.05 meters. The 1 meter height represents an attacker using the `skypie` where it could be mounted on a rover, put on a table, or be sitting out the window of a car. The 3.05 meter height represents drone flight, but could also estimate a second story of a building or use of any elevated platform. At both heights, the `BluBarry` package on the `skypie` is used to evaluate BLE attack capabilities at different distances. The research questions posit how far from a target the attacker can be using this lightweight, low-cost equipment and the `skyport` attacker web platform. `BluBarry` first evaluates the RX capability

to collect BLE advertisement packets which consist of information such as MAC addresses, device name, flags, and RSSI data. `BluBarry` then tests the TX capabilities by attempting to connect to the device, and, if successful (which is dependent on how the device manufacturer setup the device) enumerates the characteristics and their values. These characteristics contain information such as device name, specific model of that device, manufacturer, and current settings (such as in pilot studies where a Playbulb's characteristics were read and the current light level was shown). This information found in the advertisement packet (or referred to as a beacon) is vital for pattern-of-life analysis. An attacker can collect how many devices are active at a location, what the device is, and track when its no longer in use, etc. This gives the attacker opportunity to estimate the number of people at a location or hypothesize who might be in the building based on the devices present.

5.2 Device Orientation

Pilot studies were run to determine if the orientation of the target devices affected the RSSI. Beacons containing the RSSI were collected for each device for a 1 minute scan in each orientation. Figure 28 shows the device orientations in the x, y, and z axes, with the Nonin device as the example. It was found that the S340 and Nonin devices have the strongest RSSIs in the x orientation with an average difference of at least 5 RSSIs weaker in the other axes. The Masimo device was found to have the strongest RSSI in the z axis, but as the difference in averages was less than 3 RSSI, for continuity sake, the experiments were run in the x axis.

Figure 28: Orientations for Target Devices

5.3 Range

While the research hypothesized BLE collection out to 600 meters, in reality, the maximum range was 450 meters. At distances beyond 450 meters, no beacons were collected from any of the devices. Therefore, only nine collection points will be referenced, and the experiments will be compared up to 450 meters, though data may only be shown to the maximum range for that device. At each of the nine collection points from 50 meters to 450 meters, the `BluBarry` package iterated 30 times to collect advertisement packets, which resulted in about seven minutes for a `BluBarry` thread to run. This collects the minimum information necessary for pattern-of-life analysis, and collects as many RSSIs as possible in order to attempt a more accurate measure of approximate distance from the target. If the device allows such a connection, the `skypie` automatically attempts to connect and enumerate the characteristics and values. This probe is executed against each BLE device discovered in the scan.

5.3.1 Experiment One: Beacon Collection

Figures 29, 30, and 31 show the expected general trend for the RSSI values is downward, showing that the signal is weaker as the distance increases. The data is shown in box-and-whisker plots, where the minimum and maximum are shown as the whiskers and the box represents the 1st and 3rd quartiles. The median can be seen as the line inside the box, with the x denoting the average. Outliers are represented as dots outside the box. While the signal weakens as the distance between the target and attack platform increases, this strength, as long as the device is able to be detected, is not relevant in order to collect pattern-of-life data.

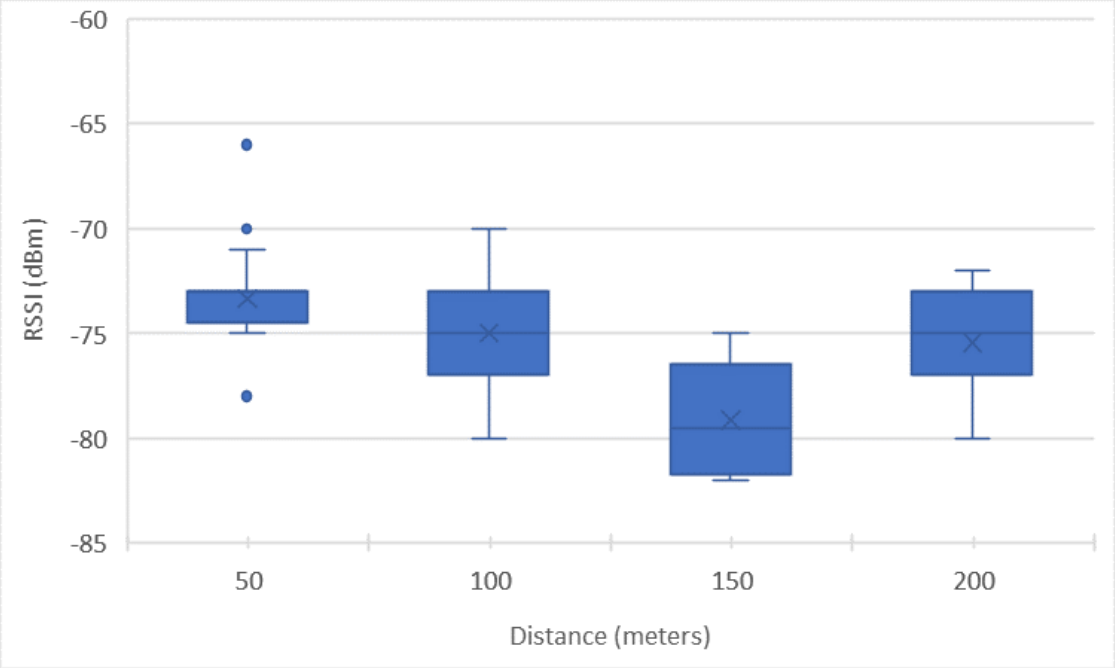


Figure 29: RSSIs for the Masimo Device for Experiment One

The data collected for the Masimo device (Figure 29) for the first experiment shows the weakest trend downwards. It is possible that the downward trend is subtle from 50 to 100 to 200 and that the 150 meter data collection is a product of ground reflection off the fresnel zone (Section 2.5), which may attenuate the signal. It is

feasible that the Masimo RSSI collection may have been more accurate in the z axis where its RSSI was the strongest. However, in a realistic environment, the attacker would have no control over the orientation of the device, and this research proves BLE beacons can be collected for the Masimo device up to 200 meters at an elevation of 1 meter. Figures 30 and 31 show the collected RSSI data for the Nonin and S340 devices, respectively, and have a stronger trend downwards to show that as the distance increases, the RSSIs decrease.

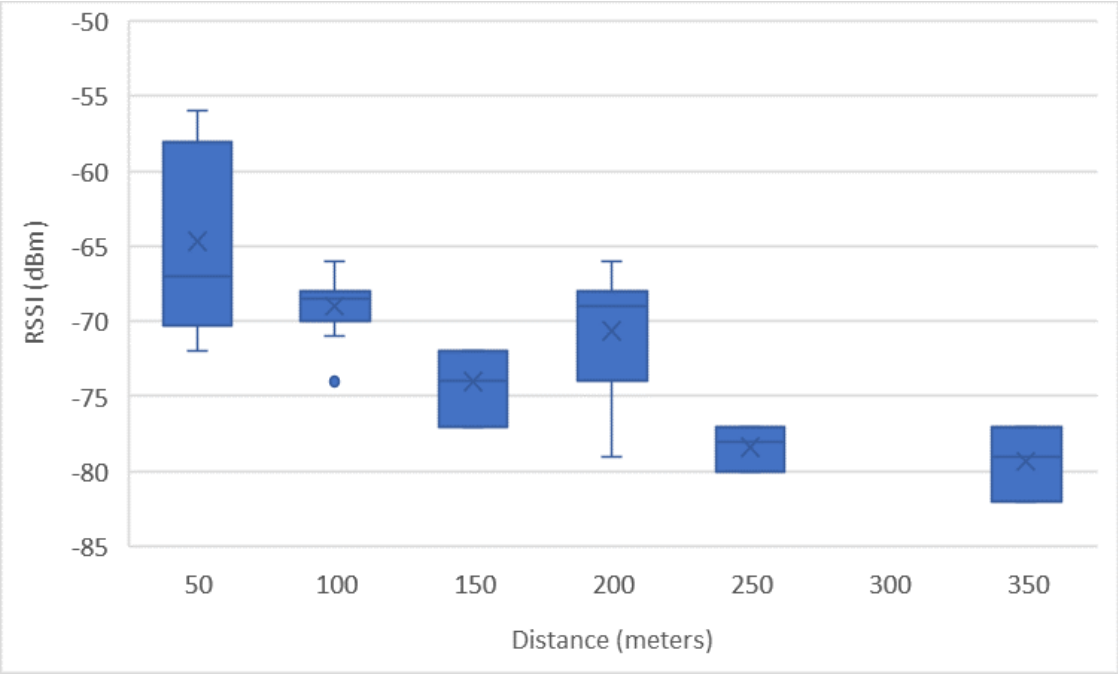


Figure 30: RSSIs for the Nonin Device for Experiment One

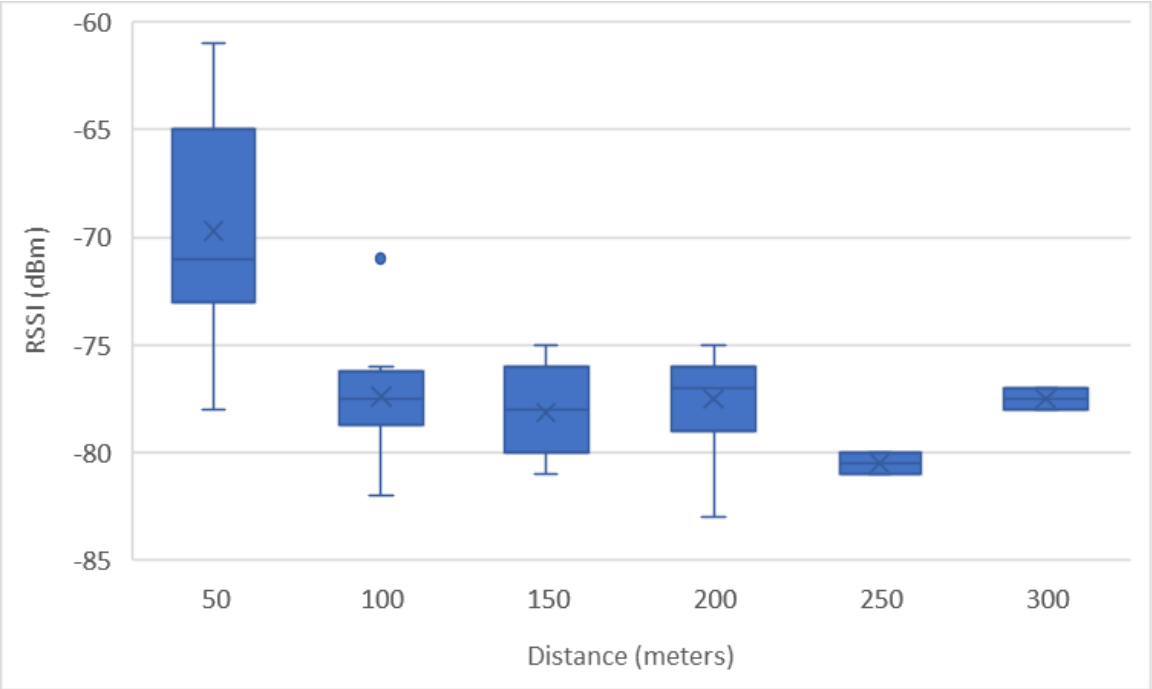


Figure 31: RSSIs for the S340 Device for Experiment One

Table 6 summarizes what the maximum distance to collect beacons was for each device, where the success means that at least one beacon was collected at a location. Failure means that no beacons were collected. While 600 meters was the goal, 450 meters was the maximum distance for collected data over the two experiments. Therefore, only nine reference points will be referenced, where the 1st represents 50 meters and the 9th represents 450 meters. The Nonin device did not collect beacons at 300 meters, but did at 350 meters. It is intuitive that the beacons should have also been collected at 300 meters, and that there was some sort of interference or particularities of the environment at the time of the 300 meter collection for the Nonin device.

Figure 32 shows the overall downward trend of the three devices as the distance increases, correlating to the RSSI becoming weaker as distance increases. Expected RSSIs for the two experiments are calculated in Section 5.4.

Table 6: Beacon Success for Experiment One

	Beacons Collected		
	Experiment One		
	Masimo	**Nonin**	**S340**
Success	4	6	6
Failure	5	3	3
Max Success Distance (meters)	200	350	300

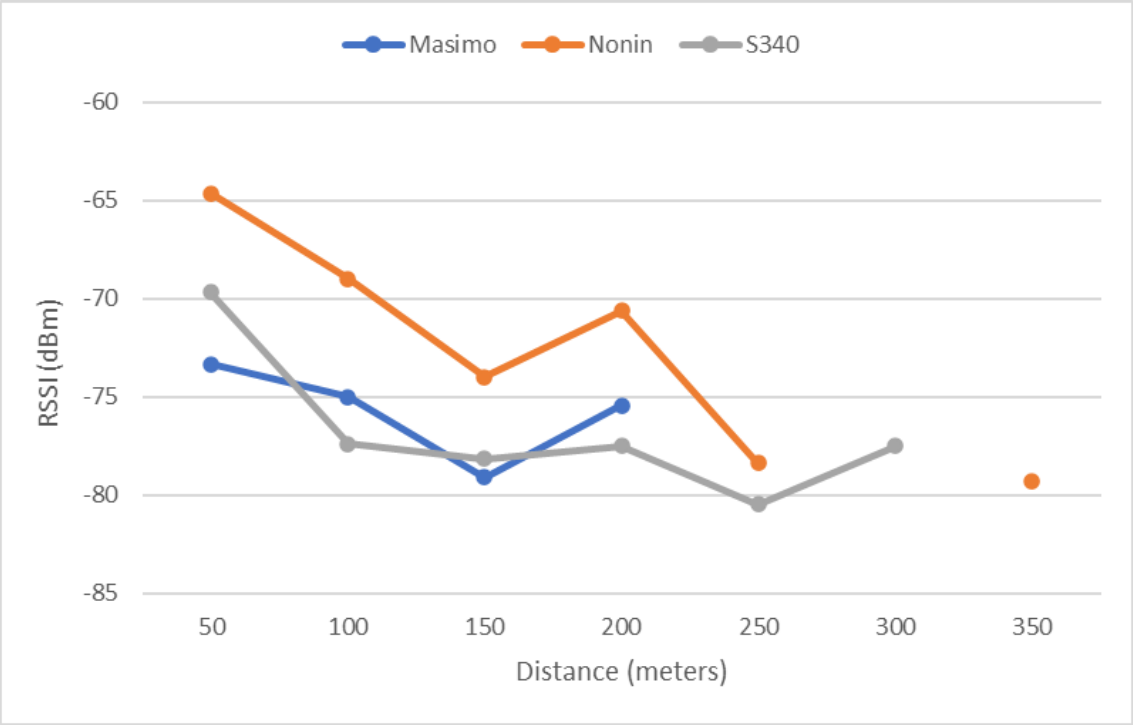


Figure 32: Average RSSIs for Experiment One

5.3.2 Experiment One: Device Connection

It was found that for the Masimo and S340 devices, connection was not possible, regardless of the distance. This is due to how the manufacture has set up connections to the device, and may require a certain type of device, such as a phone application, to secure a connection. The Nonin, however, did allow for connections using the

`BluBarry` package. Characteristics and their respective values were read at the distances of 50, 100, 150, and 200 meters, which is summarized in Table 7. At 250 meters, `BluBarry` was no longer able to obtain this data. This is conjectured to be because the distance was too far to maintain a reliable connection.

Table 7: Connection Success for Experiment One

	Characteristics & Values Read		
	Experiment One		
	Masimo	**Nonin**	**S340**
Success	0	4	0
Failure	9	5	9
Max Success Distance (meters)	N/A	200	N/A

5.3.3 Experiment Two: Beacon Collection

The experiment was rerun with the `skypie` at a height of 3.05 meters to simulate being mounted on a drone. Nine locations were again used for experiment two from 50 meters to 450 meters to scan and attempt connection to the targets. The same nine devices were also evaluated up to 600 meters, but as no signals for any of the three devices were detected, the data evaluated stops at 450 meters. Figures 33, 34, and 35 again show the overall downward trend for the devices, demonstrating that the received signal is weaker as the distance increases between the attack platform and medical IoT targets.

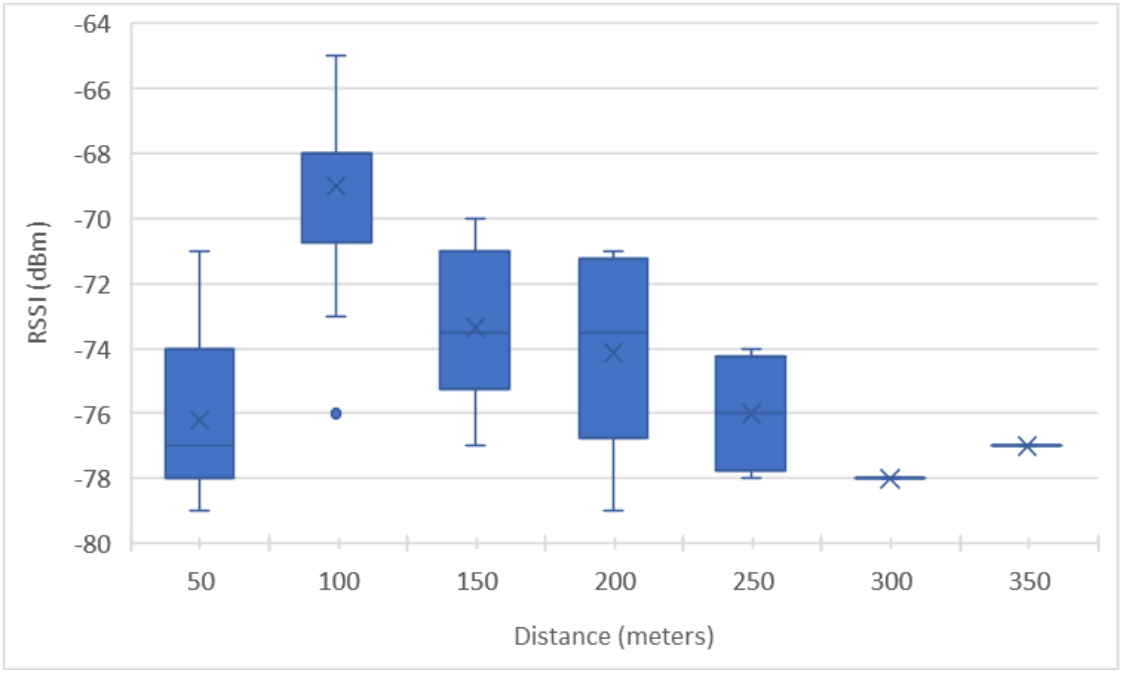


Figure 33: RSSIs for the Masimo Device for Experiment Two

The data collected for the Masimo device (Figure 33) for the second experiment shows a steeper downward trend than in the first. However, the 50 meter collection point shows a weaker RSSI average than the other distances up to 300 meters. While the cause of this is unknown, it is possible that the 50 meter collection point beacons are as readily detected at the 3.05 meter elevation due to the ellipsis associated with the fresnel zone. The Nonin RSSI connection data shown in Figure 34 shows a lower 50 meter value, similar to the Masimo device. There is also not a linear decline in RSSIs as the distance increases, but the trend is downward. Figure 35 shows a subtle, but steady decline in RSSIs from the 50 meter collection point to 450 meters for the S340.

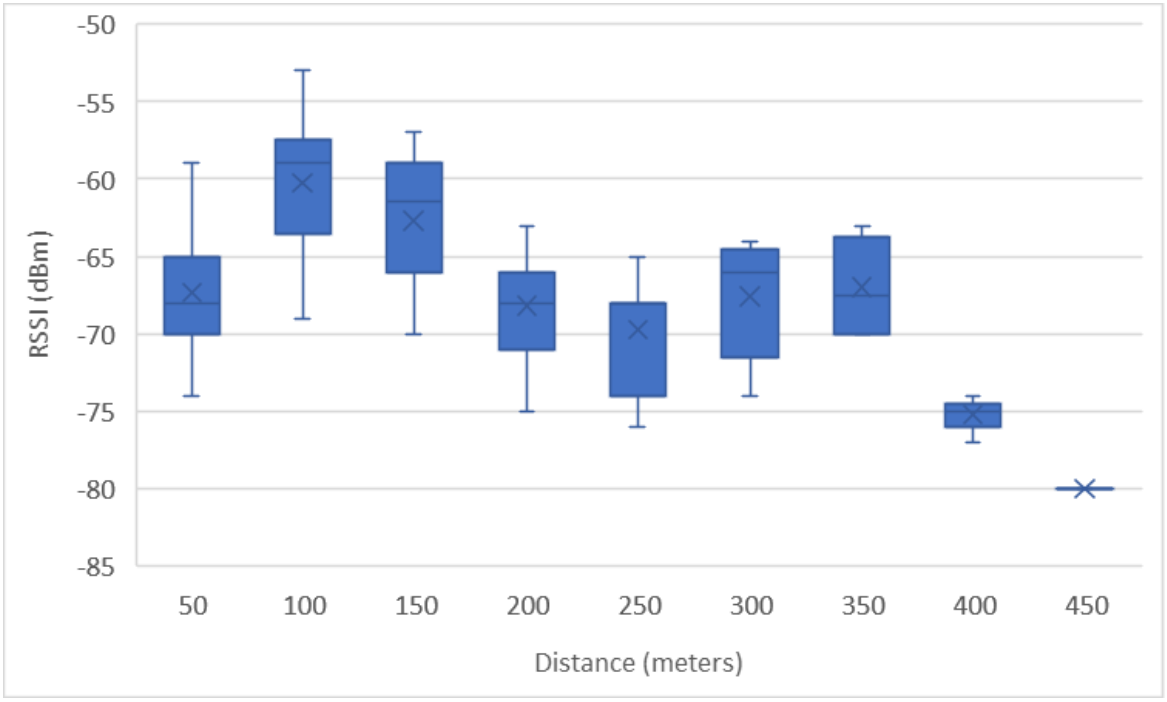


Figure 34: RSSIs for the Nonin Device for Experiment Two

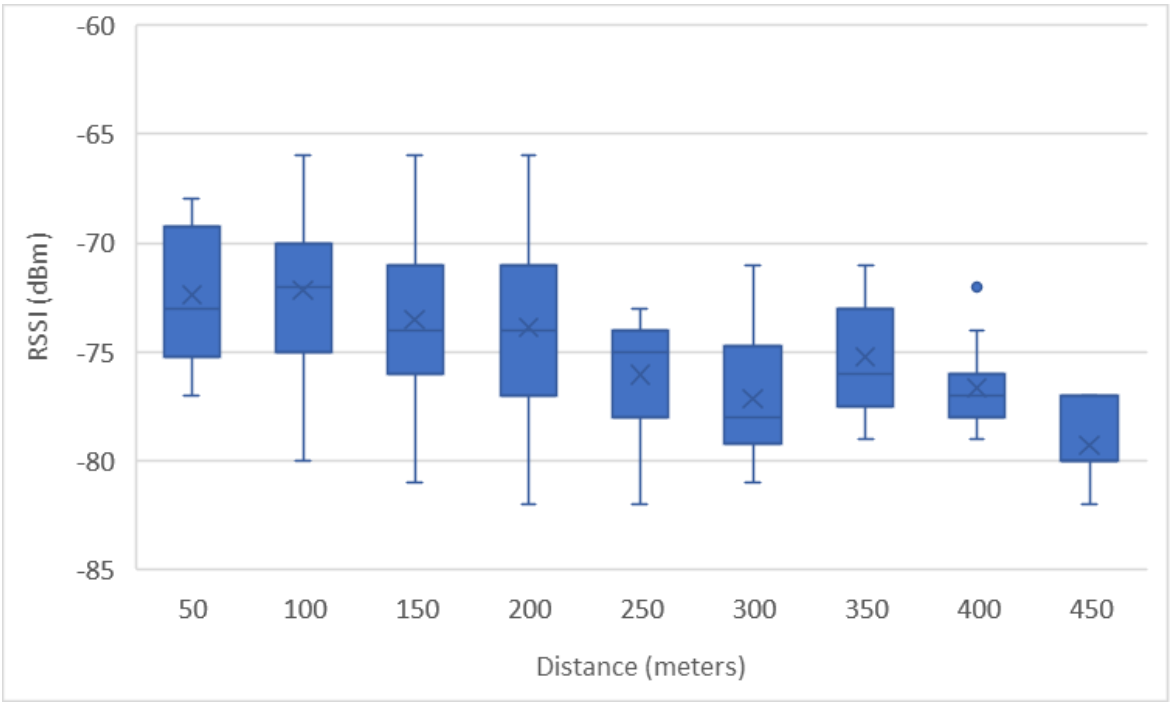


Figure 35: RSSIs for the S340 Device for Experiment Two

Table 8 summarizes whether or not a beacon was detected for the device at the specified distance. The detection distance for beacons for the Masimo was increased from 200 meters for the Masimo device to 350 meters from experiment one to experiment two, for the Nonin from 350 to 450 meters, and from 300 to 450 meters for the S340. Overall, Figure 36 shows the average RSSIs for the IoT devices over the collected distances for experiment two.

Table 8: Beacon Success for Experiment Two

	Beacons Collected		
	Experiment Two		
	Masimo	**Nonin**	**S340**
Success	7	9	9
Failure	2	0	0
Max Success Distance (meters)	350	450	450

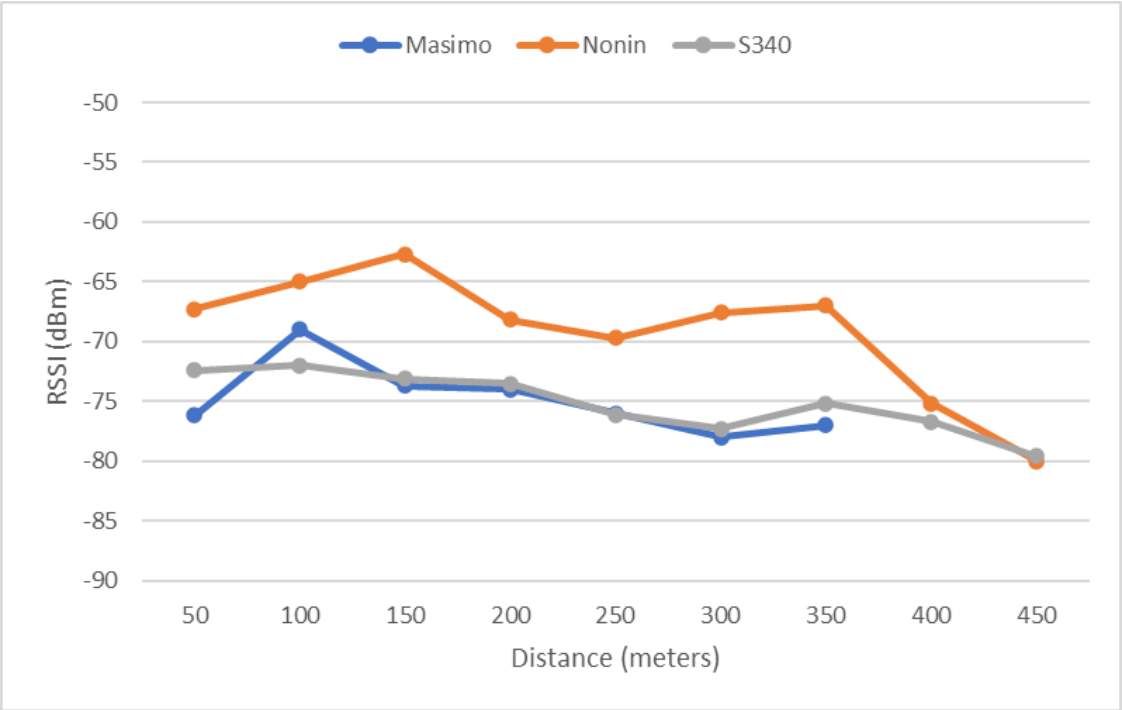


Figure 36: Average RSSIs for Experiment Two

5.3.4 Experiment Two: Device Connection

At the 3.05 meters collection height, it was found that the distance of connection had increased from 200 meters to 350 meters for the Nonin device, shown in Table 9. An analysis of the effect of height is discussed in Section 5.4.

Table 9: Connection Success for Experiment Two

	Characteristics & Values Read		
	Experiment Two		
	Masimo	**Nonin**	**S340**
Success	0	7	0
Failure	9	2	9
Max Success Distance (meters)	N/A	350	N/A

5.3.5 Range Beacons and Connections Summarized

Table 10 shows the number of collection points at which beacons were able to be collected for the device, and Table 11 shows whether or not a device was connectable, and the max distance at which the connection and enumeration were successful. If the device is connected, the attacker can attempt to write over characteristic values using the `skyport` web interface.

Table 10: Beacon Success for Experiments One and Two

	Beacons Collected					
	Experiment One			**Experiment Two**		
	Masimo	**Nonin**	**S340**	**Masimo**	**Nonin**	**S340**
Success	4	6	6	7	9	9
Failure	5	3	3	2	0	0
Max Success Distance (meters)	200	350	300	350	450	450

Table 11: Connection Success for Experiments One and Two

	Characteristics & Values Read					
	Experiment One			**Experiment Two**		
	Masimo	**Nonin**	**S340**	**Masimo**	**Nonin**	**S340**
Success	0	4	0	0	7	0
Failure	9	5	9	9	2	9
Max Success Distance (meters)	N/A	200	N/A	N/A	350	N/A

5.4 Expected RSSIs

The RSSIs collected for these experiments can be compared against theoretical RSSIs to determine the accuracy of the actual versus expected. While it is known that RSSI is not overly accurate [46], it is still useful for distance estimation. Section 2.5 discusses signal propagation theory and the associated equations that will be used for calculations. (1) is an expression to calculate the expected RSSI. It involves a path loss variable which is computed by either (2) or (3). (2) assumes Free Space Path Loss (FSPL) and (3) incorporates Ground Path Loss (GPL). Both equations are used to calculate the expected RSSIs for the experiments and compared to the actual RSSIs to determine which is a valid assumption given the experiment elevation.

Figures 37 and 38 plot the expected RSSI using the FSPL formula, the expected RSSI using the GPL formula, and the actual data for the devices in both experiments. The TxPower (P_{tx}) for the S340 and Nonin devices were found both via a `bluetoothctl` scan and confirmed using an Ubertooth. The S340 advertised a TxPower of 0 dB and Nonin of 3 dB. The dBi gain of the antenna (G_{tx}) was estimated to be 3 dBi, and the receiver antenna advertises a gain of 18 dBi. The Masimo device had the value as private, so the TxPower is unknown, and is thus omitted from the remaining analysis. However, given that this device has a shorter range in both experiments one and two, it is reasonable to assume it is below 0 dB as per the

TxPower ranges allowed by the BLE specification discussed in Section 2.5.

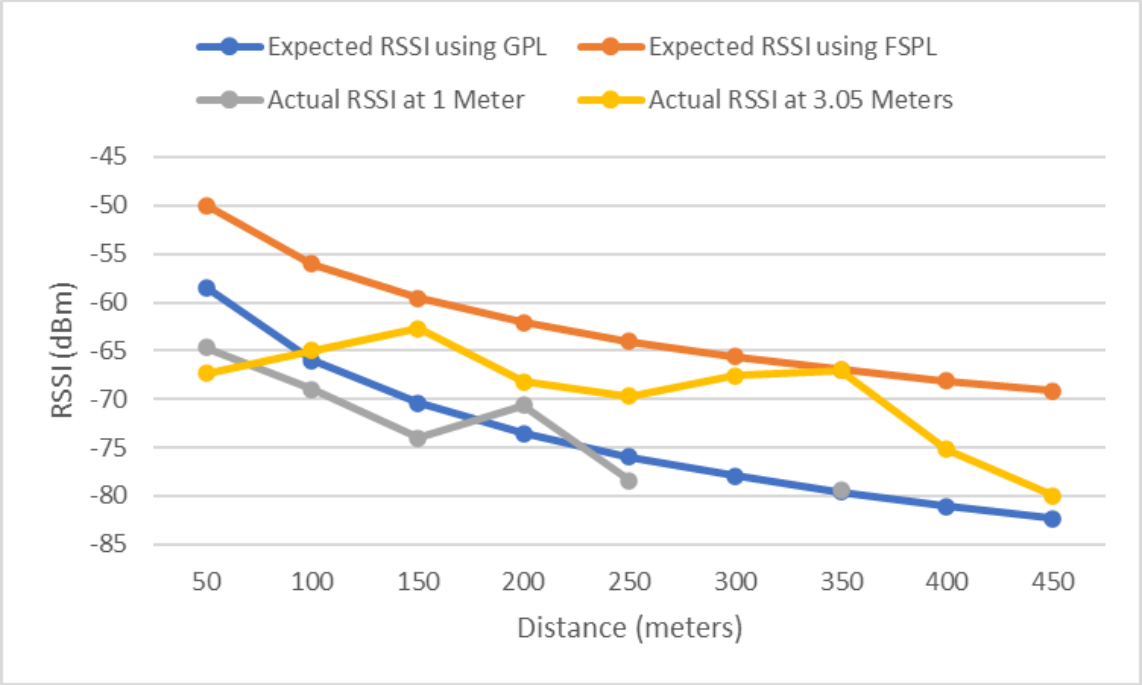


Figure 37: Expected versus Actual RSSIs for Nonin using (2) and (3)

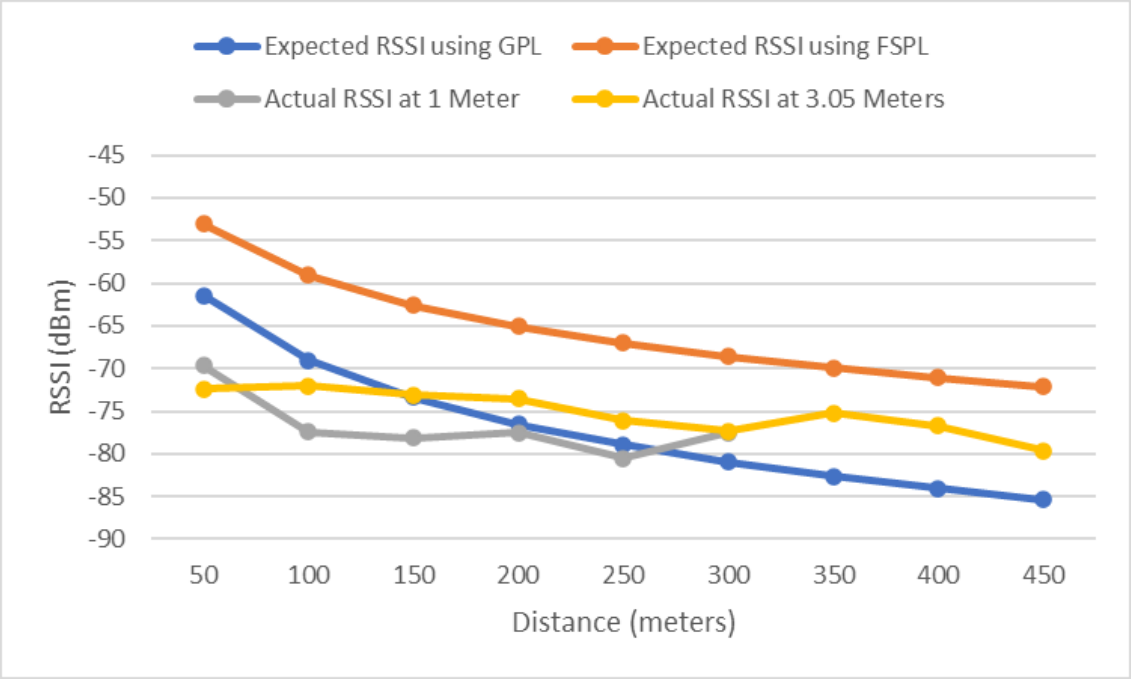


Figure 38: Expected versus Actual RSSIs for S340 using (2) and (3)

It can be seen that the values of the actual data for experiment one at 1 meter resemble the GPL formula (3). This can be explained by the Link Budget Formula (4) discussed in Section 2.5. This formula is used to determine the height at which an object must be to avoid ground interference in its cone of radio frequency. Using (1), it can be seen in Table 12 that at a distance (d) of 50 meters and a frequency (f) of 2.4 GHz, the radius (r) of the cone is 1.25 meters. This shows that experiment one at an elevation of only 1 meter already has potential ground interference, so it is reasonable that the actual data would align with (3) which incorporates ground contribution versus the FSPL formula (2). At a height of 1 meter, the `skypie` could obtain data without attenuation due to the ground only at 30 meters from the target. Table 12 shows the radius needed to avoid interference given a certain distance.

Table 12: Link Budget Fresnel Zone Calculations

Distance (meters)	50	100	150	200	250	300	350	400	450	500
Radius (meters)	1.25	1.77	2.16	2.50	2.79	3.06	3.31	3.53	3.75	3.95

Figures 37 and 38 also show that for experiment two which was run with the `skypie` at a 3.05 meter elevation, the data for each device corresponds more closely with the FSPL formula. The max range for beacon collection is extended from 350 meters in experiment one to 450 meters in experiment two. The connection, reading of characteristics, and potential for characteristic alteration by the attacker is extended from 200 meters to 350 meters. This information is summarized in Table 11.

The S340 and Nonin devices are evaluated against themselves at each distance for the 1 meter and 3.05 meter elevations in order to determine if there is a significant difference for a device at the two elevations. A null hypothesis (H_0) is formed that there is no difference between the device RSSIs at 1 meter and at 3.05 meters in height, with an alternate hypothesis (H_A) that there is a difference. The Wilcoxon/Kruskal-Wallis Rank Sums Test (referred to as Wilcoxon) is used in the JMP statistical software [52].

This test is useful for determining if there is a statistical difference between two data sets when normality cannot be assumed (therefore a nonparametric test) and when the sample numbers available are not the same.

The results of this test are displayed in Table 13 for the Nonin device. A box plot of the RSSI data being compared is shown in Figure 39. The p value is the probability of finding the results attached to the H_0 postulated [53], and is evaluated at a 95% confidence rate as signified by the α of 0.05 in the table. At a distance of 50 meters, the H_0 cannot be rejected, and the RSSIs are evaluated to be the same between the two experiments. A reason for why this might be has been discussed in Section 5.3. The remainder of the distances where beacons were collected in both experiments for the Nonin are able to reject the H_0 according to the results of the Wilcoxon test. The RSSI values collected in the second experiment with the `skypie` at an elevation of 3.05 meters are stronger RSSI values. The statistical analysis performed shows that at 100 meters or greater, the elevation of 3.05 meters results in better RSSI values.

Table 13: Wilcoxon Test for Nonin

Distance (meters)	**Sample Size***	p **value**	α	**Reject** H_o
50	18, 23	0.1922	0.05	No
100	20, 21	<0.0001	0.05	Yes
150	7, 16	<0.0001	0.05	Yes
200	24, 18	0.0339	0.05	Yes
250	5, 7	0.0025	0.05	Yes
350	3, 4	0.0238	0.05	Yes

*sample size for experiment one is listed first, then for experiment two

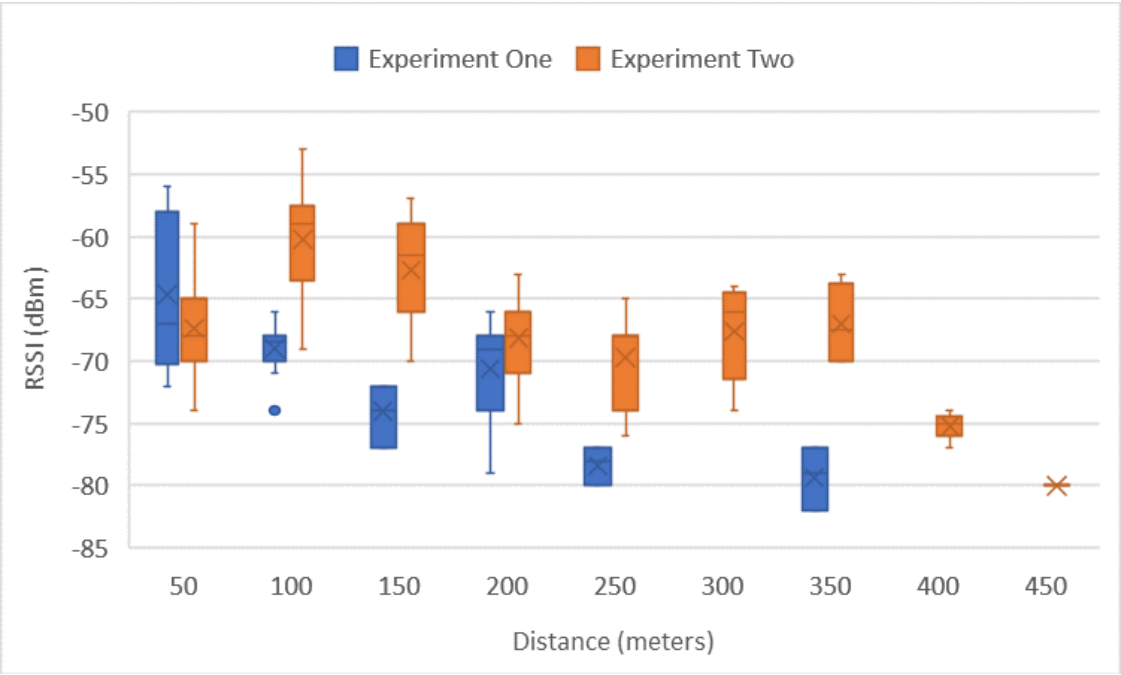


Figure 39: Nonin RSSI Data from Experiments One and Two

Table 14 displays the results from the Wilcoxon for the S340 device, and a box plot of the S340 RSSI data collected in experiments one and two can be seen in Figure 40. It is determined that, with a 95% acceptance rate, the H_0 can be rejected and that there is a difference in the RSSI values collected at 100-250 meters. At 50 meters and 300 meters, H_0 cannot be rejected. It is conjectured that the 50 meter rejection is due to the fresnel zone as discussed for the Nonin device. The 300 meter evaluation is only operating on two sample sizes for experiment one, and thus a longer collection setting may result in more collection points to show this distance also can reject H_0.

Table 14: Wilcoxon Test for S340

Distance (meters)	**Sample Size***	p **value**	α	**Reject** H_o
50	81, 16	0.0864	0.05	No
100	12, 62	<0.0001	0.05	Yes
150	19, 39	<0.0001	0.05	Yes
200	34, 23	<0.0001	0.05	Yes
250	2, 24	0.0154	0.05	Yes
300	2, 14	0.9083	0.05	No

*sample size for experiment one is listed first, then for experiment two

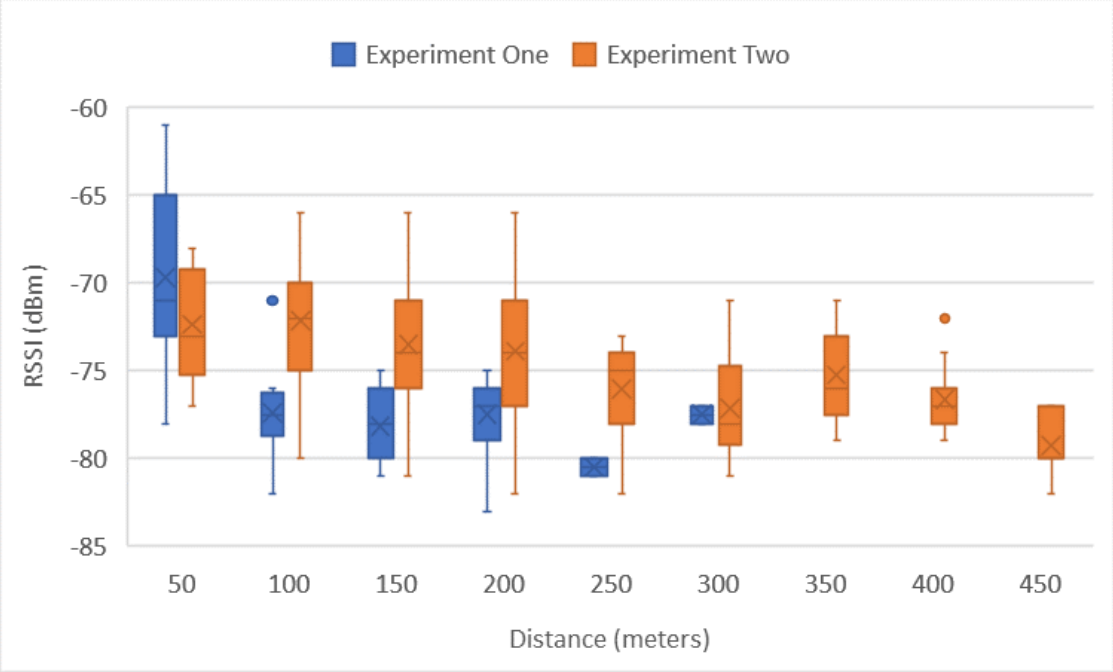


Figure 40: S340 RSSI Data from Experiments One and Two

Similarly, Figure 38 shows how close the RSSI averages are at 50 and 300 meters, and thus these results seen visually are confirmed statistically. It can be concluded that, in both of the experiments ran on the Nonin and S340, the RSSIs collected at an elevation of 3.05 are statistically different than those at 1 meter. These higher RSSI values correspond to stronger signal strength, and the elevation at 3.05 meters affords at least an extra 100 meters in beacon collection distance.

The expected RSSIs using the FSPL formula for the Nonin and S340 devices is summarized in Table 15 and 16, respectively. For experiment two, it can be seen that there is wide variance among the samples collected at each location for the collected RSSIs, with an average mean error of -6.7988 for the Nonin and -9.7136 for the S340. While the RSSI could be used for distance estimation to the target, this high error results in inaccurate predictions and would be usable only in unobstructed environments. The `skypie` platform already incorporates a more accurate measure of distance estimation in the geolocation predictions established by Bramlette in previous works [1]. This mechanism would need to be modified to collect BLE beacons instead of Wi-Fi beacons when using the `BluBarry` package, but would result in geolocation results proven in Bramlette's research.

Experiments one and two demonstrate the potential for collection and alteration on a BLE target. While the `skypie` being elevated to 3.05 meters allows for a further range, 1 meter still allows for 350 meters. Elevation at 3.05 meters causes a 100 meter increase in distance for collection. There may be different circumstances in which the attacker may want to avoid drone use (or perhaps use the drone at a low height) and use a rover or car, for example, and this experiment demonstrates the maximum ranges they can expect to intercept signals or cause effects. It also demonstrate the range at which these and similar devices are vulnerable to potential adversarial collection and injection.

Table 15: Nonin Expected RSSIs for Experiment Two

TxPower	**Device**	**Distance (meters)**	**Measured Mean (dBm)**	**Measured Median (dBm)**	**Expected (dBm)**	**Mean Error**	**Median Error**	**Standard Deviation**	**Variance**
3	Nonin	50	-67.3	-68.0	-50.031	-17.269	-17.969	4.108	16.874
3	Nonin	100	-65.0	-59.0	-56.052	-8.948	-2.948	3.820	14.590
3	Nonin	150	-62.7	-61.5	-59.574	-3.126	-1.926	4.127	17.029
3	Nonin	200	-68.2	-68.0	-62.073	-6.094	-5.927	3.240	10.500
3	Nonin	250	-69.7	-68.0	-64.011	-5.689	-3.989	3.861	14.905
3	Nonin	300	-67.6	-66.0	-65.594	-2.006	-0.406	4.037	16.300
3	Nonin	350	-67.0	-67.5	-66.933	-0.067	-0.567	3.347	11.200
3	Nonin	400	-75.2	-75.0	-68.093	-7.107	-6.907	1.095	1.200
3	Nonin	450	-80.0	-80.0	-69.116	-10.884	-10.884	N/A	N/A
					Averages	**-6.8**	**-5.7247**	**3.4543**	**12.8247**

Table 16: S340 Expected RSSIs for Experiment Two

TxPower	**Device**	**Distance (meters)**	**Measured Mean (dBm)**	**Measured Median (dBm)**	**Expected (dBm)**	**Mean Error**	**Median Error**	**Standard Deviation**	**Variance**
0	S340	50	-72.4	-73.0	-53.031	-19.369	-19.969	3.204	10.268
0	S340	100	-72.0	-72.0	-59.052	-12.948	-12.948	3.360	11.288
0	S340	150	-73.1	-74.0	-62.574	-10.526	-11.426	4.084	16.677
0	S340	200	-73.5	-73.0	-65.073	-8.427	-7.927	3.317	11.000
0	S340	250	-76.1	-75.0	-67.011	-9.089	-7.989	2.376	5.645
0	S340	300	-77.3	-78.0	-68.594	-8.706	-9.406	2.878	8.286
0	S340	350	-75.2	-76.0	-69.933	-5.267	-6.067	2.603	6.773
0	S340	400	-76.7	-77.0	-71.093	-5.607	-5.907	1.673	2.799
0	S340	450	-79.6	-80.0	-72.116	-7.484	-7.884	1.667	2.778
					Averages	**-9.714**	**-9.947**	**2.796**	**7.390**

VI. Conclusions

6.1 Overview

This chapter summarizes the research and results throughout development and experimental runs. Section 6.2 concludes the findings of this research. The contributions and significance are discussed in Section 6.3. Potential future work is then posited in Section 6.4, and a summary is provided in Section 6.5.

The `skypie` v3 is proven to be successful in beacon collection vital for pattern-of-life detection out to 350 meters at 1 meter elevation. A second experiment is run to determine the success of the `skypie` at an elevation of 3.05 meters, where the `BluBarry` collection is successful out to over a quarter of a mile (450 meters). This work is successful in maintaining the design goals set forth in the original `skypie` blueprint [1] for the prototype to be lightweight, under 1 kg, and low-cost, under $500. This allows for the `skypie` v3 to be light enough to mount on most drones, and also portable enough to use at the attacker's leisure on other potential mediums such as on a rover, in a car, or on a table. This research demonstrates what a motivated threat actor could do with low monetary resources and commercially available products. The research goal of upgrading to the `skypie` v3 with BLE CNA abilities is met. This study was successful in answering the following research questions:

1. How close does an attacker need to be to collect BLE data from a target using lightweight, low-cost equipment?

2. Can pattern-of-life data be collected at 600 meters?

3. At what distance does pattern-of-life collection become infeasible?

4. Can the attacker also interact with the target from that distance?

5. How is the range of the `skypie` affected at a height of 1 meter versus 3.05 meters?

6.2 Research Conclusions

The first research question of how close an attacker must be to collect BLE data is answerable in conjunction with questions two, three, and five. The hypothesis that BLE collection can be done out to 600 meters was proven to be unattainable given the current hardware and methodology used. However, it is determined that pattern-of-life collection is possible out to 450 meters at a 3.05 meter elevation and 350 meters at a 1 meter elevation. This demonstrates that the elevation does have an effect on the distance at which BLE signals can be collected, which is explained in Section 2.5 and analyzed in Chapter V. To answer research question four, the results show that the attacker can interact with the devices at a distance of 100-150 meters from the maximum for pattern-of-life collection, up to the distances of 200 meters for an elevation of 1 meter and 1.75 times that distance at 3.05 meters for a range of 350 meters, where the attacker can connect to and enumerate the current device settings. The attacker is then able to attempt to overwrite the current characteristics for a CNA.

In conducting experiments with varied elevations for the `skypie`, it is determined that the Free Space Path Loss formula (2) is a better model for path loss for distance estimation using RSSIs at an elevation of just over 3 meters, versus the Ground Path Loss equation (3) for a height of 1 meter. Additionally, it is determined that, using the Wilcoxon/Kruskal-Wallis Rank Sums Test, there is a statistical difference in the RSSIs collected at 3.05 meters versus 1 meter for distances 100-350 meters for the Nonin device, with an exception of 50 meters, and 100-250 meters, with an exception at 50 meters and 300 meters for the S340 device. The results of this analysis lead to

the conclusion that the 3.05 meter elevation is better for both stronger RSSI signals and longer distances for BLE pattern-of-life data and device interaction. The average mean error for the Nonin device RSSIs is -6.7988 and -9.7136 for the S340, further supporting the conclusion that RSSI is useful for general distance estimations in unobstructed environments, but not accurate enough to pinpoint the target location.

Though the three devices chosen for the experiments are all BLE fingertip pulse oximeters, this research shows the variance that comes from manufacturer programming. While they have different distances for detection and only one allows full enumeration, this point illustrates the data leakage that a device that is designed for close use can have effects even out to a quarter of a mile. It indicates that an attacker could sniff this pattern-of-life data, undetected, and determine what devices are in use at the time and posit who might be home. The `skypie` was also able to fully enumerate the characteristics on one of the devices. Long et al. shows that one of the devices can have its active connection sniffed using an Ubertooth, seeing the data passed from the Masimo device to the phone application in cleartext [38].

6.3 Research Contributions/Significance

As discussed in Section 1.7, this research has several contributions to research on airborne wireless attack, medical IoT devices, and BLE pattern-of-life collection. The `skypie`'s hardware is upgraded and the software is modified to feature BLE attack capabilities. BLE C2 capabilities are added to the `skyport` attacker interface, and the data is collected, sorted, and displayed for the attacker. The prototype is evaluated at different distances and elevations with receive only and transmit and receive toolsets.

With the increasing focus on IoT integration into the medical field, military operations, home use, and more, it is important to realize the data leakage that comes

with these IoT protocols. `skypie` v3 is now capable of:

- Geolocation of targets
- Wi-Fi beacon collection
- Wi-Fi network attack and enumeration
- BLE beacon collection to catalog BLE devices
- Probing of BLE devices for characteristics and values
- BLE device attack - writing arbitrary values to BLE characteristics on remote devices

The collection of beacons allows for an attacker to infer pattern-of-life data, such as what devices are in use and at what location. This information can be used to determine who is home, how many people are home, when the particular device is no longer in use, and what type of devices the consumer uses. In the case of the Masimo device, the data was sent in cleartext, meaning that with an Ubertooth adapter, the attacker could sniff the data. Health data such as heart rate and pulse reading could be read by an attacker, compromising a user's private information. The data leakage, particularly in a contested environment or emergency environment that BATDOK was intended for [24], could prove more detrimental to its users. An enemy could use the pattern-of-life data to estimate how many people are in a building and might be injured by how many medical devices are in use; she could go further as to sniff the actual health data for the soldiers transmitting across the devices and attempt to send malicious packets. This would provide a major military advantage for the attacker.

The Nonin device allowed the attacker to connect to and fully probe the device to get current characteristics and values. Though the characteristics were not writable

per manufacturer setting, it is possible for other devices with less diligent manufacturers to be modified. This research briefly discussed the MiPow Playbulb used in pilot studies, whose characteristics were able to be modified using `BluBarry` to turn dim and brighten the lightbulb or turn it off, and to change the name of the device. In the context of a smart home, the attacker could enumerate beyond pattern-of-life collection and have physical effects such as this.

6.4 Future Work

The `skypie` v3 has added in BLE pattern-of-life collection and showed proof-of-concept for connecting to and modifying device characteristics. However, there are multiple areas where the platform could be further developed.

1. The `skypie` has been developed using a Sena UD-100 adapter. As the distance out to 450 meters has been proven in this work, sniffing the traffic with an Ubertooth is also plausible. Work done in [38] shows that the user information for the Masimo device is sent in cleartext and can be sniffed passively. The researcher could outfit the `skypie` with an Ubertooth once the target is known and then gather more than just pattern-of-life data and current characteristics. This research avenue could also include more involved CNAs on the BLE devices such as those discussed in Section 2.3.2, like the ones demonstrated by Rose [10] to hack smartlocks. Injection attacks could also be developed where the attacker could act as a MITM.

2. The `skyport` hub has the capability for the attacker to control several `skypie` sensors remotely. Research could be conducted while utilizing multiple sensors to see how they interact and the realistic CNAs that can be performed utilizing the Wi-Fi and BLE capabilities.

3. The experiments were conducted against BLE v4.0-4.2 devices. As BLE v5.0 has a longer range advertised, follow-on researchers could test the distance for collection for a BLE v5.0 device.

4. The `skypie` platform could be expanded to include additional wireless protocols such as Zigbee and Z-wave. The Sena UD-100 Bluetooth adapter already includes BTC capabilities, so future work could involve developing the collection and attack options with BTC. Additionally, the `skypie` v3 used in this research focused on the BLE capabilities, but research could be done to simultaneously collect and attack BLE and Wi-Fi targets. This would require the researcher to find another lightweight antenna and Wi-Fi adapter to use in conjunction with the Yagi being used for BLE. As [2] proved, the Wi-Fi adapter with the Yagi antenna was capable of CNA out to 2200 meters, and thus it is reasonable to conclude that an adapter with a smaller antenna could still feasibly reach the 450 meter distance demonstrated in this work.

5. As mentioned at the conclusion of Section 5.4, the `skypie` framework for geolocation could be modified to incorporate BLE beacon collection. The work done in [1] has proven geolocation capabilities out to 600 meters, and as this research determines a maximum of 450 meters, the feature should work with little modification. This feature is particularly important since the findings in this research affirm that while RSSI is useful for distance estimation, it is not accurate enough to pinpoint a target. The RSSI values collected could be used for approximate navigation using the `skypie` towards the target, with the GPS coordinates to locate the destination of the proposed target.

6.5 Summary

This research demonstrated the capabilities of a poorly funded, yet highly motivated cyber attacker. A new iteration of the `skypie` attack platform was developed and the `skyport` web interface updated. The `skypie` consists of readily available, COTS components, totaling under $500. BLE beacon collection and characteristic write attack abilities were integrated into the attack platform as the `BluBarry` package, proving the the data leakage from BLE devices and attack capabilities. This research successfully demonstrates the collection capabilities out to 350 meters and 450 meters for a height of 1 meter and 3.05 meters, respectively. Additionally, the research shows the `BluBarry` package is capable of connecting to and interacting with BLE devices out to 200 meters at an elevation of 1 meter and 350 meters at a 3.05 meter elevation, which allows characteristic modification capabilities. These BLE beacons, gathered passively, can be used for pattern-of-life analysis to determine what devices are in use and how many devices are available, and allow the attacker to infer who is at the location and when.

Appendix A. `skypie` v3 Default Configuration File

```
## Skypie Config File.  Modifying this file (skypie-config) alters the behavior of the program.

# SFTP Server
[fileserver]
# Sensor's name, creates unique storage location on skyport.  Useful for multiple sensors.
name=starchy
# Credentials for the SFTP account of the sensor's name
verifier=catsLuv2WearSweaters!
# Port to connect over SFTP for uploading/downloading sensor data
sftp_port=2222
# IP/hostname to connect over SFTP for uploading/downloading sensor data
sftp_server=ftp.balllaboratories.org
# Weather files will be deleted or kept after uploading to the remote server
remove_after_upload=False

# Logging Settings
[log]
# File logging level.  You may want to set this to 'none' if you are worried about the sensor being discovered. [debug, info, warning, critical, none]
logging_level=debug
# Debug file size.  How big (kB) each file will be before split.  Smaller sizes give feedback faster, but bigger sizes are easier to manage.
logging_size=50

# Bluetooth Collection (not implemented)
```

```
[bluetooth]
# MAC of Bluetooth antenna used for collection.  Bluetooth is not
    supported.  Used as a placeholder.
bluetooth_mac=XX:XX:XX:XX:XX:XX

# Bluetooth Low Energy Collection Settings
[ble]
# mode for BLE - on or off
mode = on
# The MAC address of the BLE device of interest
ble_attack_mac = BB:AA:DD:AA:AA:AA
# The number of iterations that the heavy scan within blescan runs -
    information used to populate the ble_database
bleiterations = 30
# The default attack parameters - the attacker needs to fill in the
    information in brackets, which is displayed in the BLE tab of
    database information
blewriteparams = sudo gatttool -b [MAC] --char-write-req -a [handle] -n
    [newcharacteristicvalue]

# WiFi Collection
[wifi]
# Mode the wifi will be in.  This affects the mirror and collection
    threads [off,collect,mirror]
mode=collect
# MAC of WiFi antenna used for collection.  Currently supports only 1.
    Can use only first half to denote just manufacturerer (example: aa:
    bb:cc)
antenna_mac=00:25:22
# Collection interval in seconds
interval=30
```

```
47 # Size in mB of buffer for preferred packets (see bookmarks file).
       Oldest files will be removed when full.
48 size_bookmarks=500
49 # Size in mB of buffer for envelope data (geo, compass, and packet
       summary data)
50 size_envelopes=500
51 # Size in mB of all packets captured
52 size_raw=500
53 # Turn off collection of all packets, used to save space [on, off]
54 raw_collect=on
55 # Max size in mB of collected files
56 file_size_interval=10
57 # Raw filter (libcap format), the filter the antenna will use as the
       basis for collection. Only packets in this filter will be collected
58 raw_filter= wlan[0] == 0x80
59 # Bookmark filters (libcap format). Bookmarks are the only packets that
       are sent directly to skyport. They are a subsest of the raw packets
        collected.
60 #  Multiple filters are allowed. Seperate by a new line, be sure to
       indent each line with at least one space. Each one requires
       processing time, so it's not recommended to do more than 4.
61 bookmarks_filters=wlan.fc.type_subtype == 4
62  wlan_mgt.ssid=="DonutsRUs"
63  wlan_fc.type == 2
64  wlan.fc.type_subtype == 8
65
66 # MirrorMode
67 [mirror]
68 # The MAC of the attack platform. This device must be within range of
       the WiFi interface of the C2 machine
69 attack_mac=AA:AA:BB:BB:CC:CC
70 # The MAC of the victim.
```

```
71 target_mac=AA:AA:BB:BB:CC:CC
72 # 'All' will forward any traffic destined for the target's MAC address,
       allowing the attacker to send spoofed MAC frames. [all,attack_only]
73 forward_attackside=all
74 # [all,target_only]
75 forward_targetside=target_only
76
77 # Telemetry
78 [telemetry]
79 # [on,off] Store geo data
80 mode=on
81 # Max size in mB of telemetry data
82 size=80
83 # Length of time before data is written to a file in seconds
84 interval=42
85 # Calibration for the accelerometer/magnometer.  Adjust so that the
       bearing readings are close to 0 when the sensor is facing north.
       Min= -360, Max= 360
86 bearing_offset = -75
87
88 # Update/Transfer Management
89 [update]
90 # How often config changes are downloaded  (in seconds) from the SFTP
       server. 0 = Constant download attempts
91 download_wait= 30
92 # Time to wait (in seconds) after a data upload completes before
       initiating another.  0 = Constant upload attempts
93 upload_wait= 999
94 #  Changing to 'shutdown' notifies all operating threads they need to
       shutdown.  A gentle way to shut down.  Off is maintained when all
       the threads are done. Selfdestruct will fill the hard drive with 0's
        until the system crashes.  [on,shutdown,off,selfdestruct]
```

```
95 skypie_operation=on
96
97 # Attack Parameters
98 [attack]
99 # capture = start deauth and handshake thread to capture 4way handshake
100 # connect = connect to attack_mac AP with given password
101 # nmap = nmap connected network
102 mode = capture
103 # number of deauth packets to send
104 packets = 1
105 # MAC address of the target AP
106 attack_mac = AA:AA:BB:BB:CC:CC
107 # password to be used to connect to attack_mac AP
108 password =
109 # nmap parameters to be used
110 nmap_params = -sn -T3 192.168.43.1-254
111 # ping IP
112 ping_ip = 192.168.43.32
113 # how often to san for available APs and switch channels if needed
114 scan_interval = 30
115 # attack thread timeout msgs
116 message =
117 # Attack thread timeout variables in seconds
118 capture_timeout = 30
119 connect_timeout = 30
120 nmap_timeout = 30
```

Appendix B. `skypie` v3 blescan.py File

```
1  # This file is part of the BluBarry package and runs the BLE scan
       capabilities for the skypie.
2
3  from subprocess import PIPE, Popen
4  import time
5  import os
6  import signal
7  import json
8  import skypie.bleconnectquery
9  from threading import Timer
10 import sys
11
12 BLE_DATABASE = "data/synch/log/ble_database.txt"
13
14 """
15 Light Scan example
16 pi@raspberrypi:~ $ sudo hcitool lescan
17 Set scan parameters failed: Input/output error
18 pi@raspberrypi:~ $ sudo hciconfig hci0 down
19 pi@raspberrypi:~ $ sudo hciconfig hci0 up
20 pi@raspberrypi:~ $ sudo hcitool lescan
21 LE Scan ...
22 28:11:A5:8C:F3:A4 (unknown)
23 28:11:A5:8C:F3:A4 LE-Bose Color II SoundLink
24 DF:14:4B:8F:30:8A FitBark
25 DF:14:4B:8F:30:8A (unknown)
26 7A:FF:FD:75:6F:9E (unknown)
27 15:17:02:28:3B:8C (unknown)
28 28:11:A5:8C:F3:A4 (unknown)
29 """
```

```
30
31 # make an object to hold the data
32 class BLE_Device():
33
34     # Constructor
35     def __init__(self, mac, name='', type='', jsonDict=None):
36         if jsonDict is None:
37             # Create a new empty device
38             self.mac = mac
39             self.name = name
40             self.type = type
41             self.ad_flags = ''
42             self.rssi = []  # store as a list of tuples (RSSI)(timestamp
    )
43             self.characteristics = {}  # store as a dictionary of
    dictionaries (one per handle)
44         else:
45             # Create/import from json
46             self.mac = jsonDict['mac']
47             self.name = jsonDict['name']
48             self.type = jsonDict['type']
49             self.ad_flags = jsonDict['ad_flags']
50             self.rssi = jsonDict['rssi']  # store as a list of tuples (
    RSSI)(timestamp)
51             self.characteristics = jsonDict['characteristics'] # store
    as a dictionary of dictionaries (one per handle)
52
53     def get_rssi_avg(self):
54         rssis = [int(x[1]) for x in self.rssi]
55         if len(rssis) is 0:
56             return 0
57         return sum(rssis) / len(rssis)
```

```
58
59      def get_rssi_num(self):
60          rssis = [int(x[1]) for x in self.rssi]
61          if len(rssis) is 0:
62              return 0
63          return len(rssis)
64
65      def export_string(self):
66          return json.dumps(self, default=lambda obj: obj.__dict__)
67
68      def __str__(self): #go back and fix this so it prints the value of
    each thing
69          result = "MAC: " + self.mac + "\tName: " + self.name + "\n"
70          result += "Type: " + self.type + "\tAvg RSSI: " + str(self.
    get_rssi_avg()) + "\tRSSIs taken: " + str(self.get_rssi_num()) + "\n
    \n"
71          for handle in self.characteristics.keys():
72              result += "Handle: " + handle + "\tDescriptor: " + self.
    characteristics[handle]['handle_value_descriptor']
73              try:
74                  result += "\nCVH: " + str(self.characteristics[handle]['
    val_handle'])
75                  result += "\tCVHD: " + self.characteristics[handle]['
    char_value_descriptor'] + "\n"
76              except KeyError:
77                  result += "\n"
78          return result
79
80  def importBLEDatabase(inputFile):
81      with open(inputFile) as f:
82          data = f.readlines()
83          db = {}
```

```
84          for line in data:
85              jdevice = json.loads(line)
86              device = BLE_Device('', jsonDict=json.loads(line))
87              db[device.mac] = device
88          module_logger.info("Importing existing ble_database")
89
90
91
92
93  """
94  Intense Scan example
95  pi@raspberrypi:~ $ sudo btmgmt find
96  Discovery started
97  hci0 type 7 discovering on
98  hci0 dev_found: DF:14:4B:8F:30:8A type LE Random rssi -69 flags 0x0000
99  AD flags 0x05
100 name FitBark
101 hci0 dev_found: 56:FF:0B:D4:68:09 type LE Random rssi -75 flags 0x0004
102 AD flags 0x00
103 eir_len 28
104 hci0 type 7 discovering off
105 """
106 # number of iterations the heavy scan runs
107 HEAVY_SCAN_ITERATIONS = 30
108 def heavy_scan(ble_database, iterations=HEAVY_SCAN_ITERATIONS):
109     """
110     Description
111     :param ble_database: A dictionary of BLE_Device objects
112     :param iterations: Number of times btmgmt will be run
113     :return:
114     """
115     btmgmt1 = ['sudo', 'btmgmt', 'find'] # heavy scan command
```

```
116     for i in range(iterations):
117         time.sleep(1)
118         print("[ ] Iteration", i)
119         proc = Popen(btmgmt1, stdout=PIPE, stderr=PIPE)
120         results = proc.stdout.readlines()
121     # filling the database with the devices and their information
122         for i, line in enumerate(results): # 'i' will reference the
    index of the line
123             if 'dev_found' in line[:20].decode(): # only look in first
    20 chars
124                 # this is a line describing a found device
125                 tokens = line.decode().split()
126                 mac = tokens[2]
127                 if mac not in ble_database.keys(): # key is MAC, value
    is BLE_Device object
128                     bluetooth_type = tokens[4]
129                     if bluetooth_type == 'LE': # only consider Low
    Energy devices
130                         module_logger.info("Found new BLE MAC:".format(
    mac))
131                         ble_database[mac] = BLE_Device(mac) # if mac is
    not already in the database, create a new device to the database
132                         ble_database[mac].type = bluetooth_type
133                         ble_database[mac].rssi.append((round(time.time()
    , 1), tokens[7]))  # rssi is a list of tuples
134                         for j in range(1, 3):
135                             next_line = results[i + j].decode()
136                             tokens = next_line.split()
137                             if 'name' in next_line[:5]:
138                                 ble_database[mac].name = tokens[1]
139                             if 'AD flags' in next_line[:8]:
140                                 ble_database[mac].ad_flags = tokens[2]
```

```
141                 elif mac in ble_database.keys():
142                     ble_database[mac].rssi.append((round(time.time(), 1)
    , tokens[7])) # rssi is a list of tuples
143     print("[+] Scan Completed ")
144     return ble_database
145
146 import logging, threading
147 module_logger = logging.getLogger(__name__)
148
149 class BLEScanThread(threading.Thread):
150     """This is a thread for an instance of collection that will
    terminate only upon an update of the config file. """
151
152     def __init__(self, ble_devices):
153         super().__init__()
154         # set up thread
155         self.daemon = True
156         self.ble_devices = ble_devices
157         module_logger.info("[ ] Starting ble scan thread instance.")
158
159     def run(self):
160         # begins the scans
161         module_logger.info("Initiating light scan")
162         light_scan()
163         module_logger.info("[ ] Executing blescan; ble_database will
    populate with results.")
164         start_time = time.time()
165         heavy_scan(self.ble_devices)
166         # Probe each device in bleconnectquery
167         for device in self.ble_devices.values():
168             module_logger.info("Probing: " + device.mac)
169             skypie.bleconnectquery.ble_device_probe(device)
```

```
170         # Export database results
171         with open(BLE_DATABASE, 'w') as file:
172             for device in self.ble_devices.values():
173                 file.write("".join([device.export_string(), "\n"]))
174         # output to the attacker on the skyport
175         print("==============Database====================")
176         for device in self.ble_devices.values():
177             print(device)
178         print("==============RSSI====================")
179         for device in self.ble_devices.values():
180             print(str(device.mac) + " : " + str(device.get_rssi_avg()) +
    " dB")
181         end_time = time.time()
182         print("Total time was: ", ((end_time - start_time) / 60), end =
    ' minutes')
183
184 # number of iterations that the light scan runs
185 LIGHT_SCAN_ITERATIONS = 1;
186 def light_scan(iterations=LIGHT_SCAN_ITERATIONS):
187     """
188     Description
189     :param ble_database: A dictionary of BLE_Device objects
190     :param iterations: Number of times lescan will be run
191     :return:
192     """
193     hcilescan = ['sudo', 'timeout', '5s','stdbuf', '-oL', 'hcitool', '
    lescan'] # light scan command, runs for 5 seconds
194     down = ['sudo', 'hciconfig', 'hci0', 'down']
195     up = ['sudo', 'hciconfig', 'hci0', 'up']
196     for i in range(iterations):
197     # take the hci down then up to ensure its working properly
198         proc = Popen(down, stdout=PIPE, stderr=PIPE)
```

```
199         time.sleep(1)
200         proc = Popen(up, stdout=PIPE, stderr=PIPE)
201         time.sleep(1)
202     # send light scan information to lescan
203         logfile = open('data/synch/log/lescan.txt', 'w')
204         process = Popen(hcilescan, stdout=PIPE, stderr=PIPE)
205         for line in process.stdout:
206             sys.stdout.write(line.decode('utf-8'))
207             logfile.write(line.decode('utf-8'))
208     return logfile
```

Appendix C. `skypie` v3 bleconnectquery.py File

```
1  # This module is part of the BluBarry package.  It attempts to connect
       to the BLE device discovered in blescan.py - if allowed, it queries
       all the current characteristics then probes for the current values
       of the associated handles
2
3  from subprocess import PIPE, Popen
4  import time
5  import re
6
7  from skypie.blescan import BLE_Device
8
9  def ble_device_probe(ble_device):
10     """
11     :param ble_device: BLE_Device object
12     :return:
13     """
14   # command to attempt to connect and to query the characteristics if
       connection is allowed
15     getchars = ['sudo', 'gatttool', '-b', ble_device.mac, '--
       characteristics']
16     proc = Popen(getchars, stdout=PIPE, stderr=PIPE)
17     results = proc.stdout.readlines()
18   # pulls the results and populates the BLE database with the
       characteristics and their associated handles
19     for line in results:
20         tokens = line.decode().split()
21         handle = tokens[2].translate({ord(','): None})
22         ble_device.characteristics[handle] = {}
23         ble_device.characteristics[handle]['char_prop'] = tokens[6]
24         val_handle = tokens[11].translate({ord(','): None})
```

```
25          ble_device.characteristics[handle]['val_handle'] = val_handle
26
27      # takes the characteristics and handles discovered above and probes
        them for current values
28      for handle in ble_device.characteristics.keys():
29          readchars = ['sudo', 'gatttool', '-b', ble_device.mac, '--char-
        read', '-a', handle]
30          proc = Popen(readchars, stdout=PIPE, stderr=PIPE)
31          results = proc.stdout.readlines()
32      # grab the right side of the results, which contains the data. Ex:
        "Characteristic value/descriptor: 14 14" would just pull '14 14'
33          try:
34              value = results[0].decode().strip().split(': ')[1]
35              ble_device.characteristics[handle]['handle_value_descriptor'
        ] = value
36          except IndexError:
37              pass
38
39      # command to probe each handle
40          readchars = ['sudo', 'gatttool', '-b', ble_device.mac, '--char-
        read', '-a', ble_device.characteristics[handle]['val_handle']]
41          proc = Popen(readchars, stdout=PIPE, stderr=PIPE)
42          results = proc.stdout.readlines()
43      # grab the right side again which contains the data of interest
44          try:
45              value = results[0].decode().strip().split(': ')[1]
46              ble_device.characteristics[handle]['char_value_descriptor']
        = value
47          except IndexError:
48              pass
```

Bibliography

1. C. Bramlette. Cyber-Attack Drone Payload Development and Geolocation via Directional Antenna. Master's thesis, Air Force Institute of Technology, 2019. Accessed: 04 Mar 20 [Online]. Available: https://scholar.afit.edu/etd/2247/.

2. N. Barker. Development of a Drone-Mounted Wireless Attack Platform. Master's thesis, Air Force Institute of Technology, 2020. Accessed: 04 Mar 20 [Online]. Available: https://scholar.afit.edu/etd/3224/.

3. IMDB. *Smart House*. Accessed: 03 Jan 21 [Online]. Available: https://www.imdb.com/title/tt0192618/.

4. *Westinghouse All Electric Home*, Accessed: 04 Jan 21 [Online]. Available: https://www.techeblog.com/westinghouse-all-electric-home/.

5. A. Holst. *Forecast End-User Spending on IoT Solutions Worldwide from 2017 to 2025*, Accessed: 03 Jan 21 [Online]. Available: https://www.statista.com/statistics/976313/global-iot-market-size/.

6. K. Foote. *A Brief History of the Internet of Things*, Accessed: 04 Sep 20 [Online]. Available: https://www.dataversity.net/brief-history-internet-things/.

7. R. Raji. Smart Networks for Control. *IEEE Spectrum*, 31(6):49–55, 1994.

8. B. Lee. Embedded Internet Systems: Poised for Takeoff. *IEEE Internet Computing*, 2(3):24, 1998.

9. G. Maayan. *The IoT Rundown For 2020: Stats, Risks, and Solutions*, Accessed: 04 Sep 20 [Online]. Available: https://securitytoday.com/articles/2020/01/13/the-iot-rundown-for-2020.aspx.

10. A. Rose. Security Evaluation and Exploitation of Bluetooth Low Energy Devices. Master's thesis, Air Force Institute of Technology, 2017. Accessed: 04 Mar 20 [Online]. Available: https://apps.dtic.mil/dtic/tr/fulltext/u2/1054747.pdf.

11. S. Beyer. Pattern-of-Life Modeling Using Data Leakage in Smart Homes. Master's thesis, Air Force Institute of Technology, 2018. Accessed: 04 Mar 20 [Online]. Available: https://scholar.afit.edu/etd/1793/.

12. J. Sponas. *Things You Should Know About Bluetooth Range*, Accessed: 04 Sep 20 [Online]. Available: https://blog.nordicsemi.com/getconnected/things-you-should-know-about-bluetooth-range.

13. *Our History*, Accessed: 04 Apr 20 [Online]. Available: https://www.bluetooth.com/about-us/our-history/.

14. R. Triggs. *A Quick History of Bluetooth*, Accessed: 04 Sep 20 [Online]. Available: https://www.androidauthority.com/history-bluetooth-explained-846345/.

15. R. Heydon. *Bluetooth Low Energy: The Developer's Handbook*. Pearson, Accessed: 04 Sep 20.

16. *BLE Advertising Channels and Data Channels List*, Accessed: 15 Jun 20 [Online]. Available: https://www.rfwireless-world.com/Terminology/BLE-Advertising-channels-and-Data-channels-list.html.

17. *How Bluetooth LE Works - Link Layer*, Accessed: 15 Jun 20 [Online]. Available: https://medium.com/@zpcat/how-bluetooth-le-works-link-layer-b18475250259/.

18. SIG. Bluetooth Core Specification Version 4.2. *Specification of the Bluetooth System*, 2014.

19. *One Minute to Understand BLE Advertising Data Package*, Accessed: 15 Jun 20 [Online]. Available: https://github.com/greatscottgadgets/ubertooth/wiki/One-minute-to-understand-BLE-advertising-data-package/.

20. M. Afaneh. *Understanding SN and NESN in a BLE Link Layer Packet*, Accessed: 30 Oct 21 [Online]. Available: https://www.novelbits.io/understanding-sn-nesn-ble-link-layer-packet/.

21. J. Gutierrez Del Arroyo. Enhancing Critical Infrastructure Security Using Bluetooth Low Energy Traffic Sniffers. Master's thesis, Air Force Institute of Technology, 2017. Accessed: 04 Mar 20 [Online]. Available: https://apps.dtic.mil/sti/citations/AD1054652.

22. E. Simpson. *Getting Started with BLE Tessel*, Accessed: 15 Jun 20 [Online]. Available: https://tessel.io/blog/94736742342/getting-started-with-ble-tessel/.

23. L. Yushi, J. Fei, and Y. Hui. Study on Application Modes of Military Internet of Things (MIOT). In *2012 IEEE International Conference on Computer Science and Automation Engineering (CSAE)*, volume 3, pages 630–634. IEEE, 2012.

24. *Introducing BATDOK*, Accessed: Jan 06, 2021 [Online]. Available: https://rhsusa.com/batdok.

25. F. Johnsen, Z. Zieliski, K. Wrona, N. Suri, C. Fuchs, M. Pradhan, J. Furtak, B. Vasilache, V. Pellegrini, M. Dyk, et al. Application of IoT in Military Operations in a Smart City. In *2018 International Conference on Military Communications and Information Systems (ICMCIS)*, pages 1–8. IEEE, 2018.

26. A. Sadeghi, C. Wachsmann, and M. Waidner. Security and Privacy Challenges in Industrial Internet of Things. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2015.

27. D. Geneiatakis, I. Kounelis, R. Neisse, I. Nai-Fovino, G. Steri, and G. Baldini. Security and Privacy Issues for an IoT Based Smart Home. In *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1292–1297. IEEE, 2017.

28. D. Halperin, T. Heydt-Benjamin, B. Ransford, S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, and W. Maisel. Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 129–142. IEEE, 2008.

29. M. Rahman, B. Carbunar, and M. Banik. Fit and Vulnerable: Attacks and Defenses for a Health Monitoring Device. *arXiv:1304.5672*, 2013.

30. C. Wang, X. Guo, Y. Wang, Y. Chen, and B. Liu. Friend or Foe? Your Wearable Devices Reveal Your Personal Pin. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pages 189–200, 2016.

31. M. Ryan. Bluetooth: With Low Energy Comes Low Security. In *7th {USENIX} Workshop on Offensive Technologies ({WOOT} 13)*, 2013.

32. J. Gutierrez del Arroyo. *How Do I BLE Hacking*, Accessed: 15 Jun 20 [Online]. Available: https://www.youtube. com/watch?v=oP6sx2cObrY.

33. A. Rose, J. Del Arroyo, J. Bindewald, and B. Ramsey. BlueFinder: A Range-Finding Tool for Bluetooth Classic and Low Energy. In *Acad. Conf. Publ. Limited*, pages 303–312, 2017.

34. NIST. *CVE-2018-5383 Detail*, Accessed: 15 Jun 20 [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2018-5383.

35. E. Biham and L. Neumann. Breaking the Bluetooth Pairing–The Fixed Coordinate Invalid Curve Attack. In *International Conference on Selected Areas in Cryptography*, pages 250–273. Springer, 2019.

36. Zimperium. Don't Give Me a Brake – Xiaomi Scooter Hack Enables Dangerous Accelerations and Stops for Unsuspecting Riders, Accessed: 15 Jun 20 [Online]. Available: https://blog.zimperium.com/dont- give-me-a-brake-xiaomi-scooter-hack-enables-dangerous-accelerations-and-stops-for-unsuspecting-riders.

37. Booth, Louis, and Mayrany. IoT Penetration Testing: Hacking an Electric Scooter, Accessed: 15 Jun 20 [Online]. Available: http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1334205dswid=9293.

38. S. Long, R. Dill, and B. Mullins. Security Analysis of the Masimo MightySat: Data Leakage to a Nosy Neighbor. In *Proceedings of the 54th Hawaii International Conference on System Sciences*, page 6893.

39. *Ubertooth*, Accessed: 29 Dec 20 [Online]. Available: https://github.com/greatscottgadgets/ubertooth/.

40. J. Gutierrez del Arroyo, J. Bindewald, S. Graham, and M. Rice. Enabling Bluetooth Low Energy Auditing Through Synchronized Tracking of Multiple Connections. *International Journal of Critical Infrastructure Protection*, 18:58–70, 2017.

41. J. Huang, W. Albazrqaoe, and G. Xing. BlueID: A Practical System for Bluetooth Device Identification. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pages 2849–2857. IEEE, 2014.

42. C. Stoneff. *The Seven Steps of a Successful Cyber Attack*, Accessed: 15 Jun 20 [Online]. Available: https://resources.infosecinstitute.com/the-seven-steps-of-a-successful-cyber-attack/gref.

43. *NMAP*, Accessed: 03 Jan 21 [Online]. Available: https://nmap.org/.

44. *The Hacker Methodology*, Accessed: 22 May 20 [Online]. Available: https://null-byte.wonderhowto.com/how-to/hack-like-pro-hacker-methodology-0155167/.

45. C. Cimpanu. *New Silex Malware is Bricking IoT Devices, Has Scary Plans*, Accessed: 15 Jun 20 [Online]. Available: https://www.zdnet.com/article/new-silex-malware-is-bricking-iot-devices-has-scary-plans/.

46. V. Gao. *Proximity and RSSI*, Accessed: 15 Jun 20 [Online]. Available: https://www.bluetooth.com/blog/proximity-and-rssi/.

47. K. Frolic. *What is a Fresnel Zone?*, Accessed: 03 Jan 21 [Online]. Available: https://www.pagerpower.com/news/fresnel-zone/.

48. E. Goksel. *What is Fresnel Zone - Wireless Communication*, Accessed: 03 Jan 21 [Online]. Available: https://www.youtube.com/watch?v=u36zs0u3Xzg&ab_channel=EmrahG%C3%B6ksel.

49. Danets. *USB-Yagi TurboTenna Plug & Play 2.4 GHz Antenna*, Accessed: 03 Jan 21 [Online]. Available: http://www.danets.com/turbotenna/UsbYagi.php.

50. Parani. *Parani-UD100 Bluetooth 4.0 Class1 USB Adapter*, Accessed: 03 Jan 21 [Online]. Available: http://www.senanetworks.com/ud100-g03.html.

51. Adafruit. *Adafruit Ultimate GPS Breakout*, Accessed: 03 Jan 21 [Online]. Available: https://www.adafruit.com/product/746.

52. *Nonparametric Wilcoxon Test*, Accessed: 06 Jan 21 [Online]. Available: https://www.jmp.com/support/help/en/15.2/index.shtmlpage/jmp/example-of-the-nonparametric-wilcoxon-test.shtml.

53. *P Values*, Accessed: 06 Jan 21 [Online]. Available: https://www.statsdirect.com /help/ basics/p_values.htm.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
25–03–2021	Master's Thesis	Sep 2019 — Mar 2021

4. TITLE AND SUBTITLE

LONG-DISTANCE BLUETOOTH LOW ENERGY EXPLOITATION ON A WIRELESS ATTACK PLATFORM

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Long, Stephanie L., Capt, USAF

5d. PROJECT NUMBER

21G532A

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-21-M-058

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/RYAA
2241 Avionic Cir
WPAFB OH 45433-7765
COMM 937-713-8573
Email: Eric.Lam.3@us.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RYAA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

This work is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

In the past decade, embedded technology, known as the Internet of Things, has expanded for many uses. The smart home infrastructure has drastically grown to include networked refrigerators, lighting systems, speakers, watches, and more. This increase in the use of wireless protocols provides a larger attack surface for cyber actors than ever before. Wireless IoT traffic is susceptible for sniffing by an attacker. The attack platform `skypie` is upgraded to incorporate Bluetooth Low Energy (BLE) beacon collection for pattern-of-life data, as well as device characteristic enumeration and potential characteristic modification. This platform allows an attacker to mount the `skypie` to a medium of her choice, such as a drone, and collect BLE beacons within proximity whilst the attacker controls the prototype remotely. It is determined that the attacker can collect BLE beacons from over a quarter of a mile away at an elevation of 3.05 meters and interact with the device for characteristic enumeration up to 350 meters at the same elevation, and collect BLE beacons just under a quarter of a mile at an elevation of 1 meter, and interact up to 200 meters.

15. SUBJECT TERMS

This work is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Barry E. Mullins, AFIT/ENG
U	U	U	UU	138	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636, ext 7979; barry.mullins@afit.edu