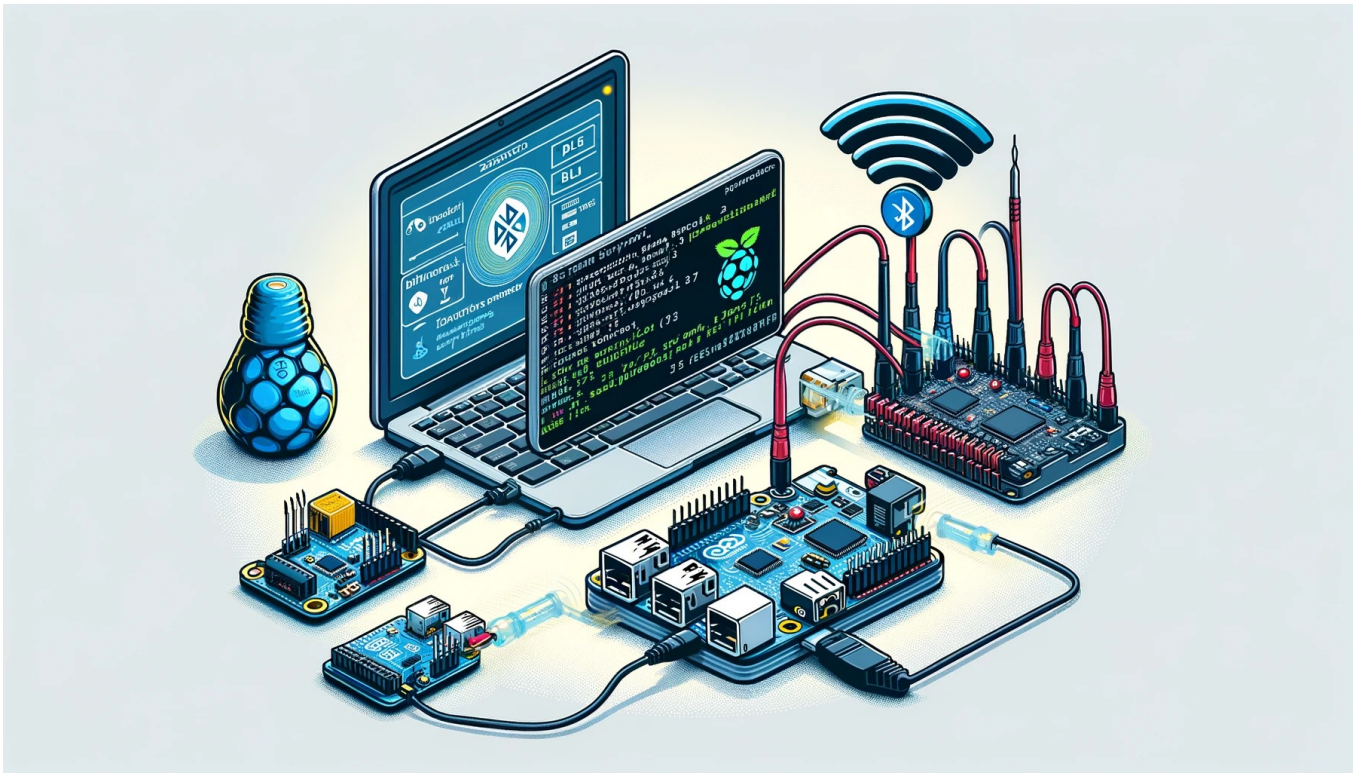


ATTACK ON BLUETOOTH LOW ENERGY



Team Members

- Abishek THAMIZHARASAN (62733)
- Megha SIVASANKAR (62775)

Content

	Page Number
• Abstract	4
• Introduction to Bluetooth Low Energy	4
• How Bluetooth Low Energy works	5
• Advantage and Disadvantages of BLE	6
• Vulnerabilities of BLE	7
• State of the art	8
• What we leaned	8
• BLE Attacks	9
• Our Setup and How it works	12
• Troubleshooting Process	13
• Challenges faced	14
• Attacks that suits our needs	15
• Attacks Performed	16
• Conclusion	26
• Reference	27

List of Figures

	Page Number
1.Connection establishment	5
2.Working of BLE	6
3.Arduino code	12
4.Raspberry pi code	13
5.Demonstration of hcitools	17
6.BtScanner	18
7.Demo of BlueRanger	19
8.BlueLog results	20
9.Spooftooph Results	21
10.Attaining Arduino information using bettercap	22
11.Scanning for other devices using bettercap	23
12.Performing MITM on smartLED using bettercap	23
13.BtleJuice proxy	24
14.Scanning for BLE device using btlejuice	25
15.Targeting Arduino using BtleJuice	25

Abstract

The "Attacks on BLE" project investigates security holes in Bluetooth Low Energy (BLE) protocols, which are essential to the Internet of Things. This paper describes the steps taken to overcome obstacles like disconnections and Bluetooth problems to create a stable communication link between an Arduino and a Raspberry Pi. The project highlights cooperative teamwork by using Raspberry Pi packages like bluz and bluepy. Future work, concentrating on BLE protocol attacks, is informed by the insights obtained. A more sophisticated understanding of securing BLE connections in the Internet of Things is made possible by documented challenges and solutions. The project's importance in the fields of IoT and cybersecurity is highlighted by this useful investigation.

Introduction to Bluetooth Low Energy

In 2010, the Bluetooth Special Interest Group (SIG) introduced Bluetooth Smart, now known as Bluetooth Low Energy (BLE), a significant advancement in wireless communication technology. The key characteristic of BLE is its remarkably low power consumption, which allows for longer battery life in connected devices. BLE offers more than just energy efficiency; it also includes easier pairing processes and cost-effective implementations, providing a range of benefits. The healthcare industry and medical devices take advantage of BLE's low-power features to constantly monitor and transmit data, improving patient care. Wearable technology such as fitness trackers and smartwatches utilize BLE to preserve battery life and maintain connectivity, guaranteeing continuous functionality. The energy-efficient connectivity of BLE is beneficial for industrial and commercial applications, such as asset tracking and beacon technology, in providing location-based services and proximity detection. BLE devices operate using advertising, scanning, and connection modes, and employ specific protocols to ensure effective data transfer while conserving power. The historical development, benefits, diverse uses, and basic operational principles of BLE are the basis for its importance in today's wireless communication environments. As BLE continues to grow and spread into various industries, its efficiency, versatility, and low-power features are making it a crucial technology in the world of wireless connectivity.

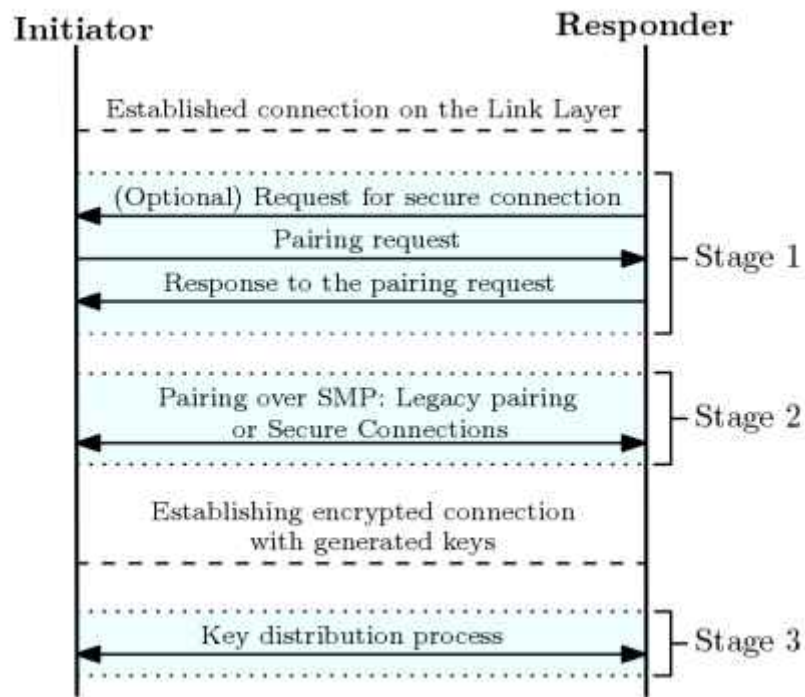


Figure 1: Connection establishment

How Bluetooth Low Energy(BLE) works

Bluetooth Low Energy (BLE) operates fundamentally differently from classic Bluetooth technology, emphasising low power consumption and efficient data transmission. Here's a breakdown of its working mechanism:

Advertising and Discovery: BLE devices operate primarily in two modes - advertising and scanning. In advertising mode, a device sends out packets of data at regular intervals to announce its presence to other devices. These packets are sent on three separate channels to reduce the likelihood of interference. Scanning devices listen for these advertising packets to discover BLE devices nearby.

Connection Establishment: Once a scanner (typically a smartphone or computer) discovers an advertising device (like a sensor or wearable), it can initiate a connection request. This connection is much more power-efficient than continuous advertising because it's based on a negotiated interval between data transfers, allowing devices to enter a low-power state when not actively communicating.

Data Transfer Using GATT: After a connection is established, data transfer takes place using the Generic Attribute Profile (GATT). GATT organises data in a structured format using "services" and "characteristics". A service is a collection of related data, and a characteristic is a specific piece of data within that collection. This arrangement allows for an organised and modular transfer of data.

Frequency Hopping Spread Spectrum: BLE uses a technique called frequency hopping spread spectrum for its radio transmissions. This means it rapidly switches between 40 different frequencies within the 2.4 GHz ISM band. This frequency hopping is done to minimise the impact of interference from other wireless devices and to enhance the security of the data being transmitted.

Efficient Power Management: One of BLE's defining features is its efficient power management. By using short data packets and enabling devices to quickly enter sleep mode between transmissions, BLE minimises energy consumption, allowing devices to operate for extended periods, even years, on tiny batteries.

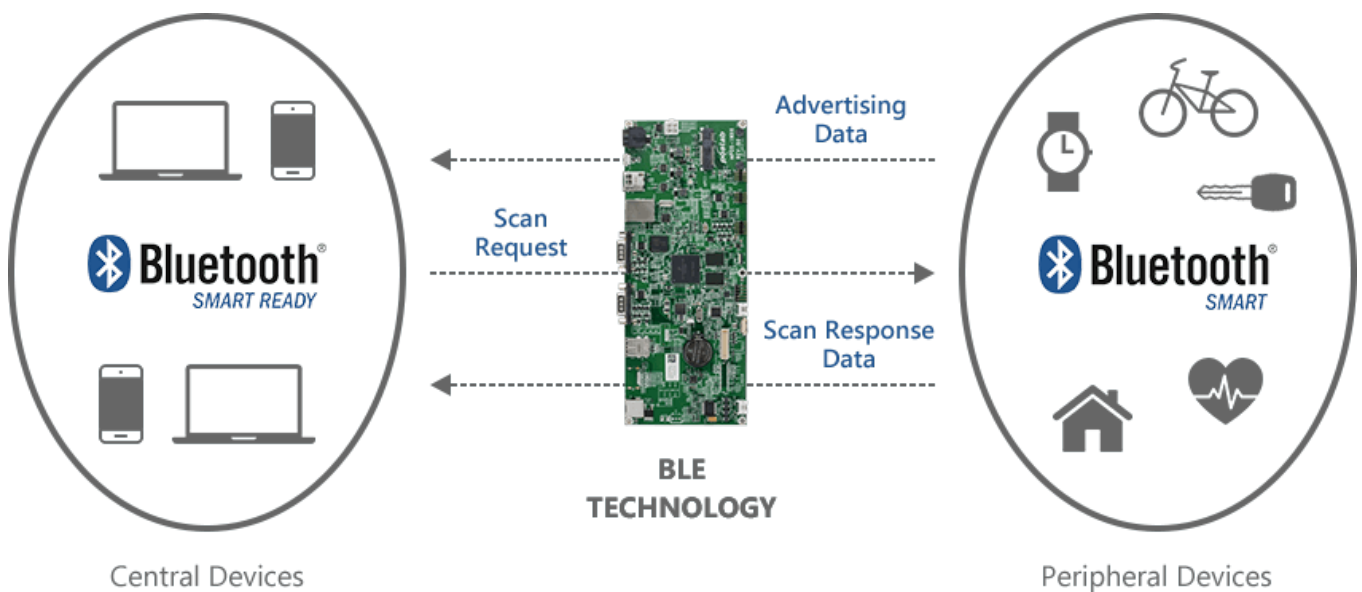


Figure 2: Working of BLE

Advantages and Disadvantages of Bluetooth Low Energy (BLE)

Advantages:

- **Low Power Consumption:** BLE's most significant advantage is its extremely low power usage, enabling devices to run for long periods on small batteries. This is particularly beneficial for wearable technology and IoT devices.
- **Widespread Device Compatibility:** BLE is supported by most modern smartphones, tablets, and computers, making it highly accessible for a wide range of applications.

- **Cost-Effective:** The technology is relatively inexpensive to implement, which helps in reducing the overall cost of BLE-enabled devices.
- **Low Latency:** BLE offers quick communication between devices, with low latency typically in the order of a few milliseconds, ideal for applications that require real-time updates.
- **Enhanced Security:** BLE incorporates robust security features, including encryption and authentication protocols, to secure data transmission against unauthorized access.

Disadvantages:

- **Limited Data Transfer Rate:** BLE is not suited for high-bandwidth applications due to its lower data transfer rate compared to classic Bluetooth and Wi-Fi.
- **Shorter Range:** While effective for short-range communication, BLE's range is limited (typically up to 100 meters in open space), which may not be sufficient for certain applications.
- **Interference from Other Devices:** Operating in the crowded 2.4 GHz band, BLE can experience interference from other wireless devices like Wi-Fi routers, leading to potential connectivity issues.
- **Security Challenges:** Despite strong security features, BLE can be vulnerable to specific types of security attacks, such as man-in-the-middle attacks, requiring ongoing vigilance in security practices.

Vulnerabilities of Bluetooth Low Energy (BLE)

- **BlueBorne:** Affects various Bluetooth devices, allowing attackers to take control and access data without pairing.
- **Bleedingbit:** Involves vulnerabilities in BLE chips that can lead to unauthorized access and control.
- **SweynTooth:** Affects BLE software, leading to security issues in IoT devices.
- **BlueFrag:** Exploits Android's Bluetooth component, potentially allowing code execution.
- **BLESA:** A vulnerability in BLE's reconnection process that enables spoofing attacks.
- **BleedingTooth:** Flaws in Linux Bluetooth subsystem, leading to potential remote code execution.
- **BlueMirror:** Involves multiple vulnerabilities in different Bluetooth technologies.
- **InjectaBLE:** Affects the security of BLE connections.
- **BrakTooth:** A set of vulnerabilities in commercial Bluetooth stacks that can lead to various attacks.
- **Pairing Mode Confusion:** Involves confusion in the pairing process of BLE.
- **BLUFFS:** Related to BLE but specific details are not provided in the summary.
- **Fixed Coordinate Invalid Curve Attack:** Involves cryptographic vulnerabilities in BLE.
- **KNOB:** A vulnerability in the Bluetooth protocol allowing interception of communications.

- **BIAS**: Exploits the Bluetooth pairing process for impersonation attacks.
- **Pairing Method Confusion**: Similar to the Pairing Mode Confusion, involves vulnerabilities in the pairing process.
- **Spectra**: Involves vulnerabilities in Wi-Fi and Bluetooth modules.
- **BLURtooth**: Exploits cross-transport key derivation in dual-mode devices.

State of the art

The comprehensive review of security and privacy in Bluetooth Low Energy (BLE) across various studies highlights the evolving landscape of BLE vulnerabilities, attack methodologies, and countermeasures. These papers collectively explore the feasibility of app-level attacks, the detection and mitigation of spoofing attacks, and the specific challenges presented by reconnection attacks like BLESa. They underscore the importance of understanding BLE protocol intricacies, the potential for downgrade attacks in key negotiations, and the broad spectrum of security threats in IoT and wearable devices contexts. Through systematic analysis, novel attack vectors such as BLE Injection-Free attacks are identified, emphasising the need for robust security frameworks. Moreover, the exploration of security vulnerabilities in Bluetooth technology, particularly within IoT applications, reveals critical insights into safeguarding against exploitation. This body of work significantly contributes to our understanding of the state of BLE security, guiding future research and development efforts to enhance the resilience of BLE-enabled devices against emerging threats.

What we learned

Evolving Security Mechanisms: How BLE's security protocols have adapted over time to address emerging threats.

- **Vulnerability Analysis:** Specific vulnerabilities within the BLE protocol and potential exploits.
- **Attack Methodologies:** Detailed exploration of various attack vectors, including spoofing, reconnection attacks, and MITM strategies.
- **Mitigation Strategies:** Recommendations and best practices for securing BLE communications.
- **Future Directions:** Insights into ongoing research areas for enhancing BLE security and privacy.

BLE Attacks

Bluetooth Low Energy (BLE) attacks refer to various security threats and exploits targeting devices that use the Bluetooth Low Energy protocol for communication. BLE is a wireless communication technology designed for short-range communication between devices, commonly used in applications like fitness trackers, smartwatches, medical devices, and IoT (Internet of Things) devices.

Here are some common types of BLE attacks:

1. Man-in-the-Middle (MITM) Attacks:

Description: Hackers intercept communication between your devices.

Prevention: Use encryption (like HTTPS) and implement authentication protocols (like SSL/TLS) to secure data transmission.

Tools Example:

- Wireshark
- Ubertooth

2. Denial-of-Service (DoS) Attacks:

Description: Floods the devices with excessive requests, causing them to crash or become unavailable.

Prevention: Implement rate limiting, firewalls, and use intrusion detection systems to filter out malicious traffic.

Tools Example:

- LOIC (Low Orbit Ion Cannon)
- Hping
- Slowloris

3. Physical Attacks:

Description: Unauthorized access or tampering with the hardware.

Prevention: Physically secure devices, limit physical access, and use tamper-resistant hardware when possible.

Tools Example: N/A (Physical attacks involve direct access and manipulation, rather than software tools.)

4. Brute Force Attacks:

Description: Repeatedly trying different combinations to gain access (e.g., cracking passwords).

Prevention: Enforce strong, unique passwords and implement account lockout policies after multiple failed attempts.

Tools Example:

- John the Ripper
- Hashcat

6. Eavesdropping:

Description: Monitoring communication to gather sensitive information.

Prevention: Use encryption and secure protocols for data transmission.

Tools Example:

- Wireshark
- GATTacker
- BLEAH

7. Software Vulnerabilities:

Description: Exploiting weaknesses in the software or firmware.

Prevention: Regularly update firmware and software to patch known vulnerabilities. Also, follow secure coding practices.

Tools Example:

- Metasploit
- Nessus
- Burp Suite

These attacks highlight the importance of implementing robust security measures in BLE-enabled devices, including encryption, secure pairing, and regular software/firmware updates to mitigate potential vulnerabilities. Security best practices and staying informed about emerging threats are crucial for maintaining the integrity of BLE-based systems.

Here some of the Types of Vulnerabilities in BLE Attacks:

1. BlueBorne:

Vulnerability Description: Exploits vulnerabilities in the Bluetooth stack to execute remote code on a device.

Preventive Measures: Keep devices updated with the latest firmware and security patches. Disable Bluetooth when not in use.

Example Tools:

- BlueBorne Scanner

2. BleedingBit:

Vulnerability Description: Involves two critical vulnerabilities in Bluetooth chips that could allow an attacker to execute arbitrary code or launch a Denial-of-Service (DoS) attack.

Preventive Measures: Implement secure pairing methods and promptly update device firmware.

Example Tools:

- Not applicable (vulnerability analysis tools may be used)

3. SweynTooth:

Vulnerability Description: A set of vulnerabilities affecting Bluetooth Low Energy (BLE) chips, allowing an attacker to trigger crashes, deadlock, or bypass security mechanisms.

Preventive Measures: Regularly update software and firmware to address known vulnerabilities. Employ secure pairing mechanisms.

Example Tools:

- Not applicable (vulnerability analysis tools may be used)

4. BtleJuice:

Vulnerability Description: A tool designed for man-in-the-middle attacks on BLE connections.

Preventive Measures: Secure BLE communications, monitor for unusual activities, and be aware of tools like BtleJuice that may exploit vulnerabilities.

Example Tools:

- BtleJuice

5. BLE-CTF (Capture the Flag):

Vulnerability Description: Represents a Capture the Flag challenge where participants are tasked with exploiting vulnerabilities in BLE implementations.

Preventive Measures: Participate in ethical hacking activities to understand potential vulnerabilities in BLE implementations.

Example Tools:

- Not applicable (This is more of a challenge than a specific vulnerability with associated tools)

6. CRACKLE:

Vulnerability Description: A tool used for exploiting BLE vulnerabilities, especially in the context of Bluetooth Smart (Bluetooth Low Energy or BLE) devices.

Preventive Measures: Employ strong encryption practices and regularly update firmware to prevent vulnerabilities exploitable by tools like CRACKLE.

Example Tools:

- CRACKLE

7. BTLJUICE:

Vulnerability Description: BTLJUICE is likely mentioned as a tool, not a vulnerability. It's used for manipulating and injecting Bluetooth traffic.

Preventive Measures: Be aware of tools like BTLJUICE, as they can potentially be used for man-in-the-middle attacks. Implement secure communication practices to mitigate the risks associated with traffic manipulation.

Example Tools:

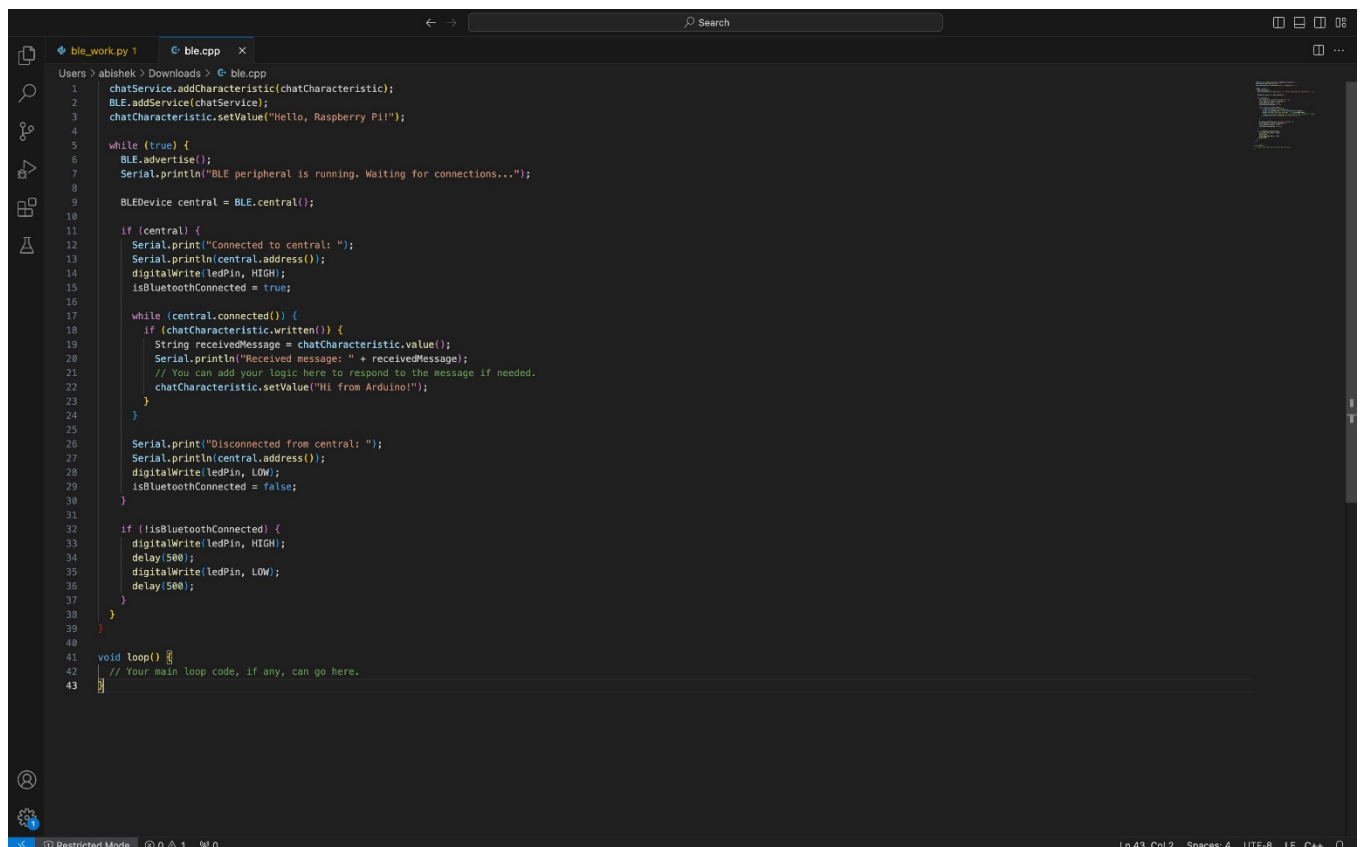
- BTLJUICE

Our Setup and How it works

In our BLE communication project, the Arduino Uno WiFi Rev 2 board uses the ArduinoBLE library to create a BLE peripheral. This library allows Arduino to advertise BLE services and characteristics, manage connections, and handle data communication. The Raspberry Pi 3 uses Python with the `pygatt` library, which provides tools to interact with BLE devices. The `pygatt` library handles the discovery of BLE devices, connection management, and data transmission.

The Arduino code advertises a BLE service with a characteristic, sending a greeting message to the Raspberry Pi and responding to received messages. The Raspberry Pi's Python script connects to this service using the Arduino's BLE address and the characteristic's UUID. It subscribes to receiving notifications from the Arduino and sends messages back, demonstrating a bidirectional communication flow between the two devices.

Arduino code:

A screenshot of an IDE window showing a C++ file named 'ble.cpp'. The code implements a BLE peripheral using the ArduinoBLE library. It starts by including the library and defining a chat service and characteristic. The main logic is in a while loop that advertises the service, waits for connections, and handles incoming messages. When a connection is established, it prints the central's address and turns on an LED. It then enters a nested while loop to handle messages: if a message is received, it prints it and sends a response 'Hi from Arduino!'. If the connection is lost, it prints a message and turns off the LED. The code also includes a setup function to initialize the serial port and a loop function for the main program flow.

```
1 chatService.addCharacteristic(chatCharacteristic);
2 BLE.addService(chatService);
3 chatCharacteristic.setValue("Hello, Raspberry Pi!");
4
5 while (true) {
6   BLE.advertise();
7   Serial.println("BLE peripheral is running. Waiting for connections...");
8
9   BLEDevice central = BLE.central();
10
11   if (central) {
12     Serial.print("Connected to central: ");
13     Serial.println(central.address());
14     digitalWrite(ledPin, HIGH);
15     isBluetoothConnected = true;
16
17     while (central.connected()) {
18       if (chatCharacteristic.written()) {
19         String receivedMessage = chatCharacteristic.value();
20         Serial.println("Received message: " + receivedMessage);
21         // You can add your logic here to respond to the message if needed.
22         chatCharacteristic.setValue("Hi from Arduino!");
23       }
24     }
25
26     Serial.print("Disconnected from central: ");
27     Serial.println(central.address());
28     digitalWrite(ledPin, LOW);
29     isBluetoothConnected = false;
30   }
31
32   if (!isBluetoothConnected) {
33     digitalWrite(ledPin, HIGH);
34     delay(500);
35     digitalWrite(ledPin, LOW);
36     delay(500);
37   }
38 }
39
40 void loop() {
41   // Your main loop code, if any, can go here.
42 }
43
```

Figure 3: Arduino code

Raspberry Pi Code:

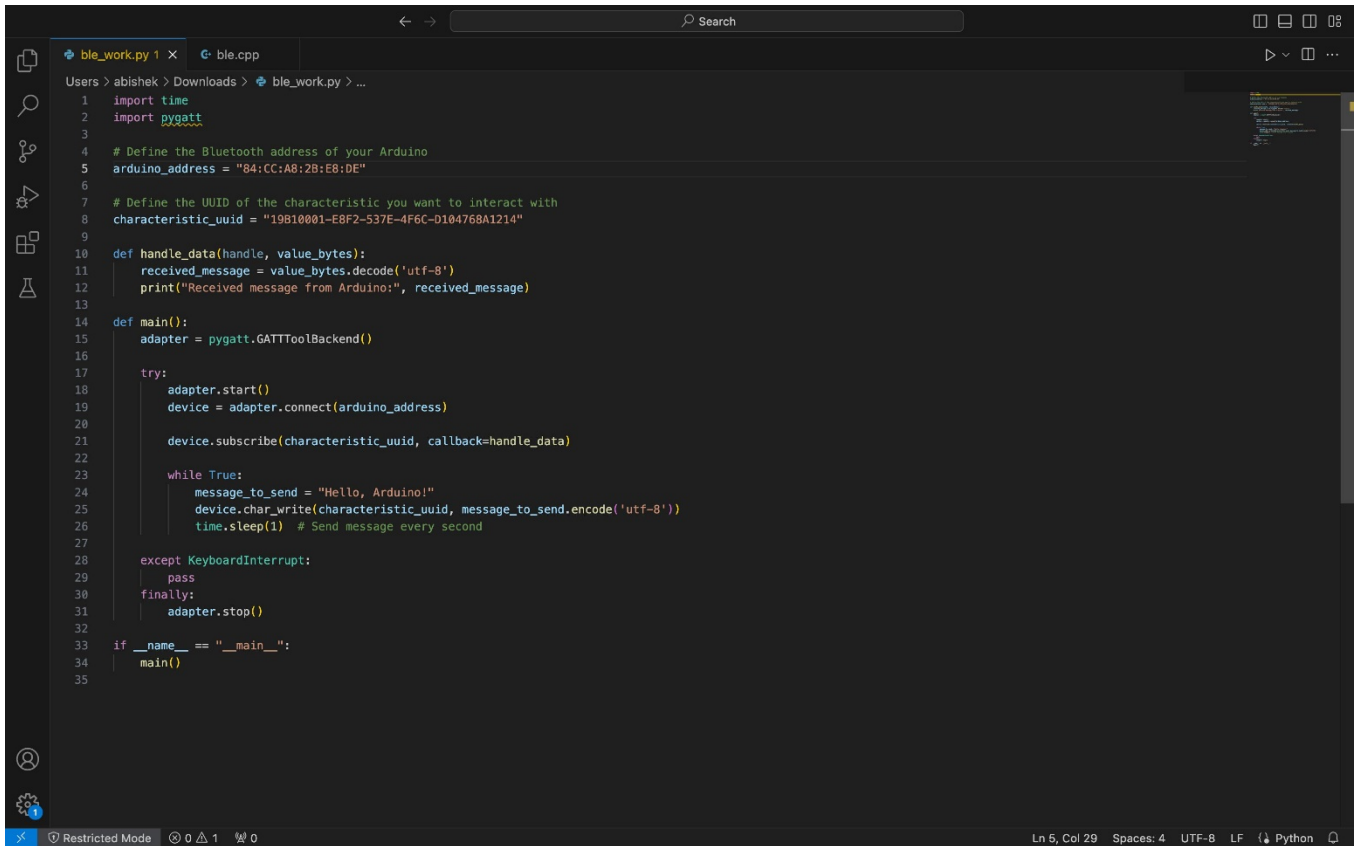
A screenshot of a code editor window with a dark theme. The editor shows a Python file named 'ble_work.py' with 35 lines of code. The code imports 'time' and 'pygatt', defines a Bluetooth address and a UUID, and implements a function 'handle_data' to process received messages. The 'main' function sets up the GATTToolBackend, connects to the specified address, subscribes to the characteristic, and sends a 'Hello, Arduino!' message every second. The code is enclosed in a try-except block to handle KeyboardInterrupt. The status bar at the bottom indicates 'Ln 5, Col 29', 'Spaces: 4', 'UTF-8', 'LF', and 'Python'.

Figure 4: raspberry pi code

In our setup's next phase, we employ a machine running Kali Linux Live, renowned for its security testing capabilities. This system is equipped with additional USB Bluetooth dongles, enhancing its ability to interact with and analyze BLE communications. This configuration is essential for executing our planned security assessment, where we aim to identify and potentially exploit vulnerabilities in the BLE communication established between the Arduino Uno WiFi Rev 2 and the Raspberry Pi 3. This setup is a standard approach in wireless security testing, allowing us to simulate attack scenarios in a controlled environment.

Troubleshooting Process

- Connection Establishment:

The initial phase involved establishing a connection between the Arduino and Raspberry Pi. This step is critical as it lays the foundation for subsequent communication and attack implementations.

- Installed Packages:

To facilitate the project, various packages were installed on the Raspberry Pi, including bluz, bluepy, pygatt, and Python 3.8. Each package serves a specific purpose in enabling BLE communication and attack functionalities.

Challenges Faced

- **Connection Issues:**

One of the initial challenges encountered was the intermittent disconnection of the established connection. Through systematic troubleshooting, we identified potential causes and implemented corrective measures to ensure a stable connection.

- **Bluetooth Disabled:**

Another obstacle surfaced when it was discovered that Bluetooth was disabled on the Arduino board. This presented an impediment to communication and required Bluetooth to proceed with the project.

- **Code Execution and Disconnection:**

During testing, the team observed instances where the code execution led to a prompt disconnection within seconds. Investigating this issue involved examining the code logic and identifying points of failure.

Arduino Code Explanation:

The provided Arduino code utilizes the Pygatt library to establish a Bluetooth Low Energy (BLE) connection with the Raspberry Pi. The code defines the Bluetooth address and UUID (Universally Unique Identifier) of the characteristic to interact with. It includes a function, `handle_data`, responsible for handling received data. Within the `main()` function, the code initializes the Pygatt backend, establishes a connection with the Arduino device using its Bluetooth address, and subscribes to the specified characteristic for incoming data. The loop continuously sends a predefined message, "Hi from Raspberry Pi!", encoded in UTF-8, to the Arduino board at one-second intervals.

Raspberry Pi Code Explanation:

Similarly, the Raspberry Pi code employs the Pygatt library to establish a BLE connection with the Arduino device. It defines the Bluetooth address and UUID of the characteristic, as well as the `handle_data` function for processing incoming data. The `main()` function initiates the Pygatt backend, connects to the Arduino device via its Bluetooth address, subscribes to the specified characteristic for incoming data, and continuously sends the message "Hi from Raspberry Pi!" encoded in UTF-8 to the Arduino at one-second intervals.

- **Restarting Computer and Traceback Error:**

Upon restarting the computer, an unexpected traceback error emerged, disrupting the workflow. Resolving this error involved revisiting the code and addressing compatibility issues between the software components.

- Conclusion:

The troubleshooting process provided valuable insights into the complexities of Bluetooth communication between the Arduino and Raspberry Pi. While encountering various challenges, each obstacle presented an opportunity to delve deeper into understanding and resolving connectivity issues, ultimately resulting in a successful and stable connection between the devices.

Attacks that suits our Project

Name of the Attack	Type of the Attack	Short Explanation	Is it Suitable?
Bettercap	MITM, Various	A versatile tool for MITM and other network protocol attacks, including Bluetooth.	Yes, but since we are using Arduino and Raspberry Pi which needs Complex command, we can only use to execute some commands.
hcitool, hciconfig	Scanning, Config	Tools for scanning and configuring Bluetooth devices.	Yes
l2ping	DoS	Useful for executing DoS attacks like BlueSmack.	No, it's not suitable because it may damage the Arduino.
Btlejack	BLE Hijacking	Specialized for hijacking BLE communications.	No
Crackle	Cracking Encryption	Used for cracking BLE encryption.	Yes, but since we are not using any encryption there is no use for this attack.
Btscanner	Scanning	A tool focused on scanning Bluetooth devices.	Yes
BT Audit	Auditing	For auditing security aspects of Bluetooth devices.	Yes
Bleah	BLE Attacks	Useful for BLE scanning and attacks, though deprecated (replaced by Bettercap).	Yes
Bluesnarfer	OBEX Attack	Attacks targeting OBEX services in Bluetooth.	No, because we are not using OBEX.

Spooftooph	Spoofing/Cloning	Automates spoofing or cloning Bluetooth device information.	Yes, but we need the information of the device we need to clone.
BtleJuice	BLE MITM	A framework designed for conducting BLE MITM attacks.	Yes
Gattacker	GATT MITM	Focuses on GATT-based Bluetooth MITM attacks.	Yes, but the tool is not supported anymore.
RedFang	Finding non-discoverable Bluetooth devices	Locates non-discoverable Bluetooth devices.	No, Since the device we are using is already discoverable.
Bluelog	Logging	A tool to log and track visible Bluetooth devices.	Yes
BlueRanger	Distance estimation	Estimates the distance to a Bluetooth device.	Yes

Attacks we performed

hcitool:

The hcitool command-line tool to perform reconnaissance on nearby Bluetooth Low Energy (BLE) devices. The process involved scanning the environment to detect and gather information about available BLE devices. Key details like device addresses (MAC), class data, and other identifying information were collected. This information is crucial for advanced attacks such as bluejacking and bluesnarfing.

The specific commands used in our experiment were:

```
$ hciconfig to display configured devices.
$ sudo hciconfig hci0 up to activate the HCI device.
$ hcitool scan for scanning nearby BLE devices.
$ l2ping <MAC> to test connectivity with a specific device.
$ sdptool browse --tree --l2cap <MAC> for detailed device information.
```



```
kali@kali: ~  
File Actions Edit View Help  
hci0: Type: Primary Bus: USB  
      BD Address: F8:9E:94:F4:30:BB ACL MTU: 1021:4 SCO MTU: 96:6  
      UP RUNNING  
      RX bytes:20925 acl:0 sco:0 events:3290 errors:0  
      TX bytes:797787 acl:0 sco:0 commands:3288 errors:0  
  
(kali@kali)-[~]  
$ hcitool scan  
Scanning ...  
    E0:B6:55:39:77:C8      MIBOX4 5847  
    1A:0D:3F:B3:89:00      MEGHA  
    22:22:7F:96:38:4B      Freebox Player POP  
    28:39:5E:41:4F:66      [TV] Samsung 6 Series (65)  
  
(kali@kali)-[~]  
$ l2ping 28:39:5E:41:4F:66  
Can't create socket: Operation not permitted  
  
(kali@kali)-[~]  
$ sudo l2ping 28:39:5E:41:4F:66  
Ping: 28:39:5E:41:4F:66 from F8:9E:94:F4:30:BB (data size 44) ...  
44 bytes from 28:39:5E:41:4F:66 id 0 time 35.92ms  
44 bytes from 28:39:5E:41:4F:66 id 1 time 11.88ms  
44 bytes from 28:39:5E:41:4F:66 id 2 time 11.77ms  
44 bytes from 28:39:5E:41:4F:66 id 3 time 7.66ms  
44 bytes from 28:39:5E:41:4F:66 id 4 time 11.90ms  
44 bytes from 28:39:5E:41:4F:66 id 5 time 11.81ms  
44 bytes from 28:39:5E:41:4F:66 id 6 time 11.75ms  
44 bytes from 28:39:5E:41:4F:66 id 7 time 11.86ms  
44 bytes from 28:39:5E:41:4F:66 id 8 time 15.91ms  
44 bytes from 28:39:5E:41:4F:66 id 9 time 11.62ms  
44 bytes from 28:39:5E:41:4F:66 id 10 time 12.02ms  
44 bytes from 28:39:5E:41:4F:66 id 11 time 11.67ms  
44 bytes from 28:39:5E:41:4F:66 id 12 time 28.06ms  
44 bytes from 28:39:5E:41:4F:66 id 13 time 11.59ms  
44 bytes from 28:39:5E:41:4F:66 id 14 time 12.19ms  
44 bytes from 28:39:5E:41:4F:66 id 15 time 11.74ms  
44 bytes from 28:39:5E:41:4F:66 id 16 time 11.86ms  
44 bytes from 28:39:5E:41:4F:66 id 17 time 15.74ms  
44 bytes from 28:39:5E:41:4F:66 id 18 time 11.80ms  
44 bytes from 28:39:5E:41:4F:66 id 19 time 16.00ms  
44 bytes from 28:39:5E:41:4F:66 id 20 time 15.93ms  
44 bytes from 28:39:5E:41:4F:66 id 21 time 11.63ms  
44 bytes from 28:39:5E:41:4F:66 id 22 time 11.77ms  
44 bytes from 28:39:5E:41:4F:66 id 23 time 11.76ms  
44 bytes from 28:39:5E:41:4F:66 id 24 time 12.00ms  
44 bytes from 28:39:5E:41:4F:66 id 25 time 11.86ms  
44 bytes from 28:39:5E:41:4F:66 id 26 time 15.85ms
```

Figure 5: Demonstration of hcitools

This setup allowed us to assess vulnerabilities in BLE devices within proximity and understand the potential risks associated with BLE communications.

BtScanner:

In our BLE security assessment, we utilized 'btscanner', a powerful tool for scanning Bluetooth devices. By executing the command '\$ sudo btscanner <MAC>', we conducted an inquiry scan to gather information about specific BLE devices. This scan provided us with detailed data, including device names, MAC addresses, and other pertinent information. This step was crucial in our analysis, as it allowed us to understand the visibility and discoverability of devices in the BLE environment, a key factor in assessing potential vulnerabilities.

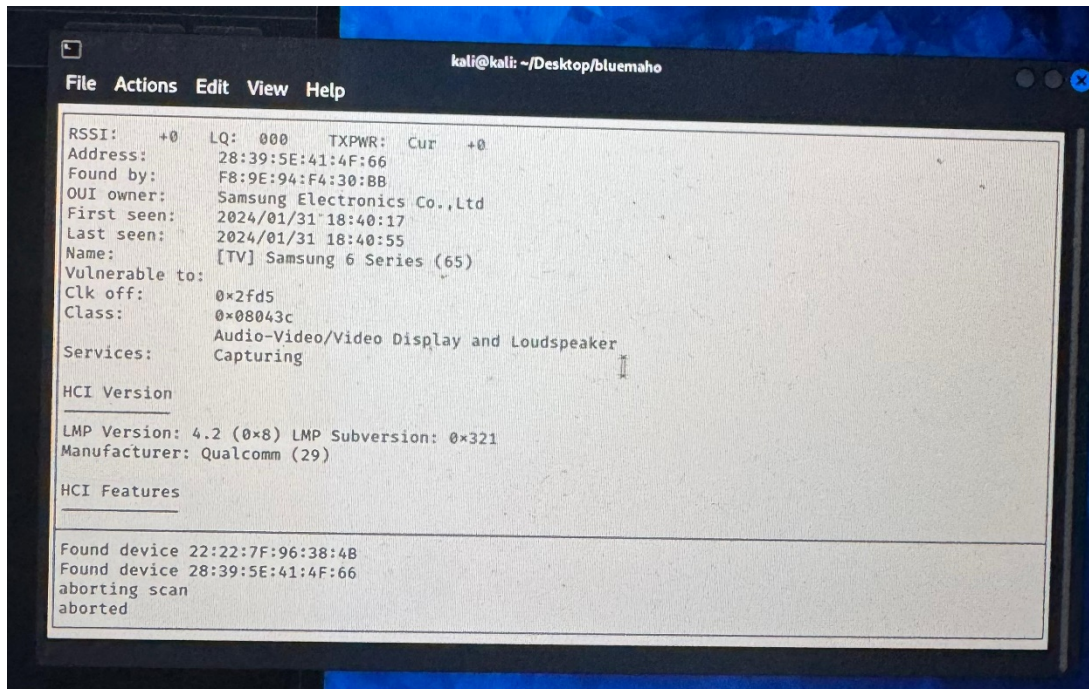


Figure 6: BtScanner

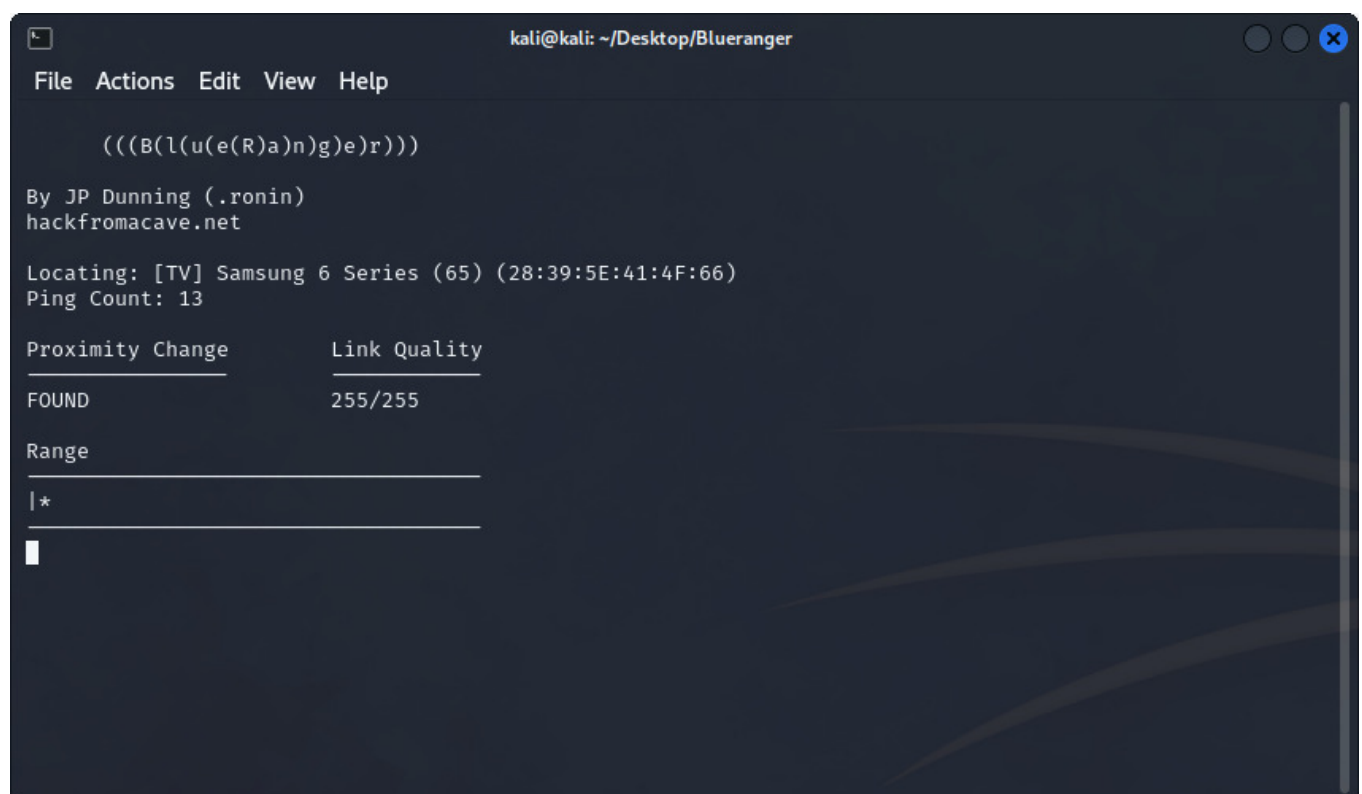
BtSnaffer and BlueJack:

In our BLE security analysis, we considered employing BtSnaffer and BlueJack attacks. These attacks typically target Android smartphones by exploiting vulnerabilities exposed through information gathered from tools like 'btscanner' and 'hcitool'. However, in our project setup, which involves an Arduino and a Raspberry Pi as target devices, we found these attacks to be unsuitable. BtSnaffer and BlueJack are more effective against devices with specific profiles and services that are commonly found in smartphones but not in our chosen hardware. This highlights the importance of selecting appropriate attack strategies based on the target device's characteristics and capabilities.

BlueRanger:

In our project, we successfully implemented the BlueRanger attack to measure the distance between our scanning device and the target BLE device. Using the command ``$ blueranger.sh hci0 <MAC>``, we were able to ascertain the proximity of the targeted device. This approach is particularly useful for understanding the physical security aspects of BLE communications, as it provides insights into how distance affects the vulnerability of BLE devices to potential attacks. This successful implementation of BlueRanger adds a crucial dimension to our security analysis, highlighting the importance of spatial considerations in BLE security.

In this attack, | - denotes the our device and * - denotes the opposite device.



```
kali@kali: ~/Desktop/Blueranger
File Actions Edit View Help

  (((B(l(u(e(R)a)n)g)e)r)))

By JP Dunning (.ronin)
hackfromacave.net

Locating: [TV] Samsung 6 Series (65) (28:39:5E:41:4F:66)
Ping Count: 13

Proximity Change      Link Quality
-----
FOUND                  255/255

Range
-----
| *

```

Figure 7: Demo of BlueRanger

Bluelog:

In our project, we successfully utilized Bluelog for Bluetooth device logging. The command ``$ sudo bluelog -i hci0 -o ~/Desktop/btdevices.log -v`` was executed to capture basic information about nearby Bluetooth devices. This command logs essential details such as device names and addresses. Furthermore, we conducted a more advanced scan using ``$ sudo bluelog -i hci0 -mnc -o ~/Desktop/btdevices2.log -v``, which provided comprehensive information about the detected devices. This advanced logging included metadata and class information, offering deeper insights into the Bluetooth devices within our proximity. These successful logging operations were crucial for our BLE security analysis, providing a rich dataset for further examination and assessment.

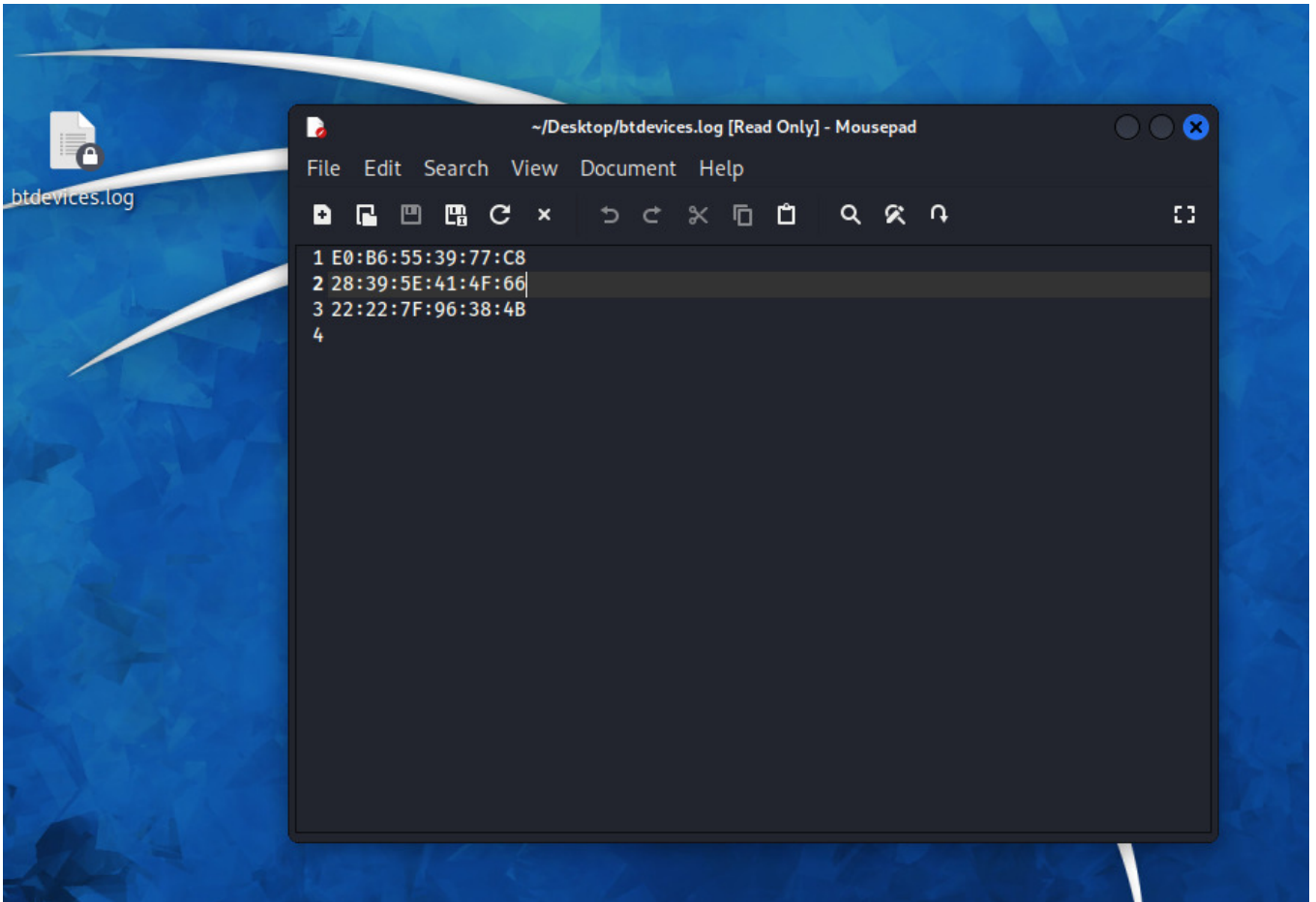


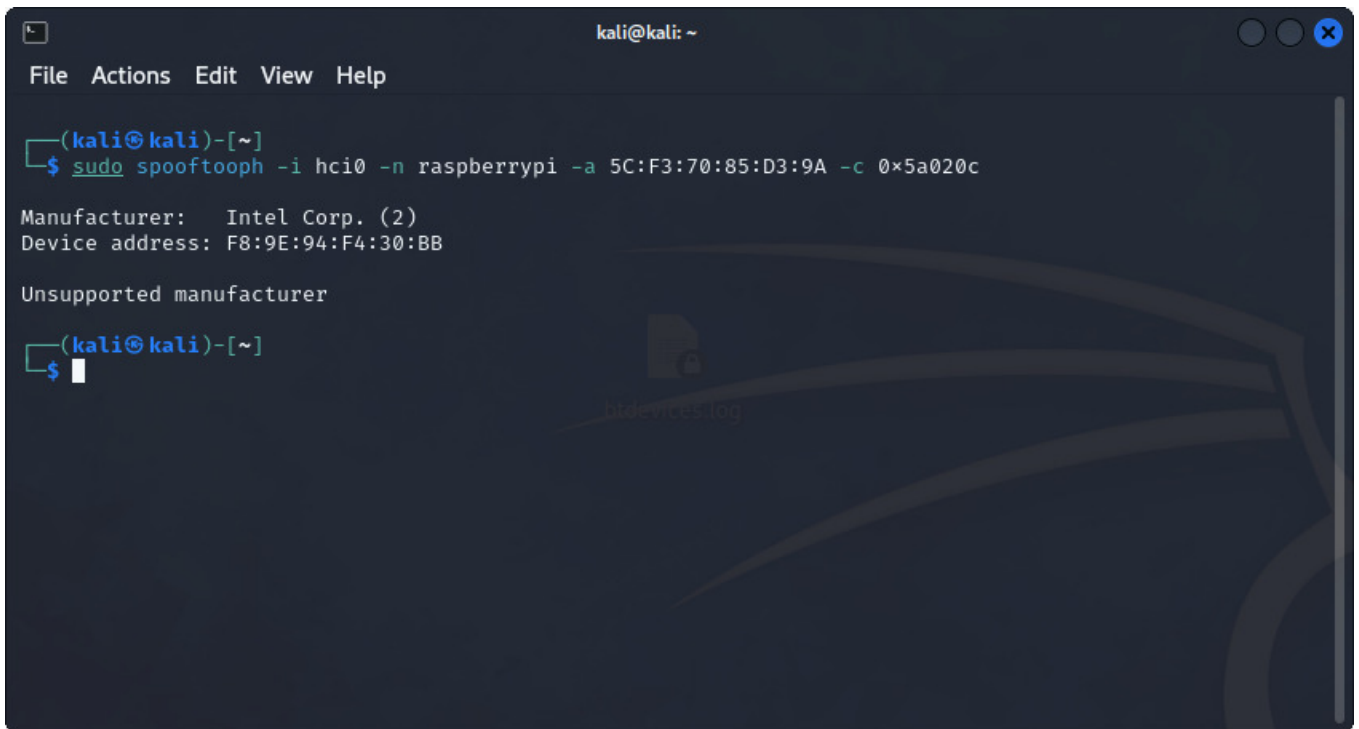
Figure 8: BlueLog Results

Spooftooph:

In our project, we conducted a successful spoofing attack using Spooftooph. The command `sudo spooftooph -i hci0 -n raspberrypi -a 5C:F3:70:85:D3:9A -c 0x5a020c` was executed to mimic the identity of a known device. This involved spoofing the name, MAC address, and class of the Raspberry Pi. However, a limitation of this attack in our context is the requirement of prior knowledge of the target device's details. While effective, this precondition of knowing specific device information beforehand limits the applicability of Spooftooph in scenarios where such information is not readily available or in more dynamic environments.

Troubleshoot:

- The dongle presented in our computer cannot be used for spoofing. So, we used an additional USB dongle, and we used that interface to perform this attack.



```
kali@kali: ~  
File Actions Edit View Help  
(kali@kali)-[~]  
$ sudo spooftooph -i hci0 -n raspberrypi -a 5C:F3:70:85:D3:9A -c 0x5a020c  
Manufacturer: Intel Corp. (2)  
Device address: F8:9E:94:F4:30:BB  
Unsupported manufacturer  
(kali@kali)-[~]  
$
```

Figure 9: Spooftooph output

Gattacker:

In our project, we intended to use Gattacker for GATT-based MITM attacks. However, we encountered a significant obstacle: the support for Gattacker has been discontinued. This limitation prevented us from installing and utilizing the tool effectively. The discontinuation of support for such tools is a common challenge in cybersecurity practices, emphasizing the need for ongoing adaptation and the search for alternative solutions. This experience underscores the dynamic nature of the field, where tools and techniques must be continually evaluated for their current relevance and effectiveness.

Bettercap:

In our project, we utilized Bettercap for BLE scanning and MITM attacks. Initially, we used **ble.recon on** to scan for nearby BLE devices and **ble.enum <MAC>** to gather detailed information about a specific device. We successfully tested a MITM attack on a Bluetooth smart LED light, demonstrating the practicality of Bettercap in manipulating BLE devices. However, our primary targets, the Arduino and Raspberry Pi, required more complex commands beyond simple on/off instructions. This limitation highlights the specificity needed in attack commands for different BLE devices. Despite this, our successful manipulation of the smart LED, where we used **ble.write <MAC> <Handle> <Command>** to toggle the light on and off, showcases Bettercap's capabilities in a real-world scenario.


```

192.168.1.0/24 > 192.168.1.104 » ble.recon 1000000 1000 ...
192.168.1.0/24 > 192.168.1.104 » ble.recon 00
192.168.1.0/24 > 192.168.1.104 » [15:47:56] [ble.device.lost] BLE device EE:26:5A:53:40:71 (Apple, Inc.) lost.
192.168.1.0/24 > 192.168.1.104 » [15:47:56] [ble.device.lost] BLE device Arduino_BLE 84:CC:A8:2B:E8:DE (Arduino) lost.
192.168.1.0/24 > 192.168.1.104 » [15:47:56] [ble.device.lost] BLE device 48:73:71:81:E4:84 (Apple, Inc.) lost.
192.168.1.0/24 > 192.168.1.104 » [15:47:56] [ble.device.lost] BLE device F2:44:3A:57:12:95 (Apple, Inc.) lost.
192.168.1.0/24 > 192.168.1.104 » [15:47:56] [ble.device.lost] BLE device 49:73:7B:7E:C8:87 (Apple, Inc.) lost.
192.168.1.0/24 > 192.168.1.104 » [15:47:56] [ble.device.lost] BLE device GBK_H618F_6B65 60:74:F4:2E:6B:65 lost.
192.168.1.0/24 > 192.168.1.104 » [15:47:56] [ble.device.lost] BLE device 49:79:B5:F3:2C:41 (Apple, Inc.) lost.
192.168.1.0/24 > 192.168.1.104 » [15:47:56] [ble.device.lost] BLE device E2:A6:3D:A5:32:F4 (Apple, Inc.) lost.
192.168.1.0/24 > 192.168.1.104 » [15:47:56] [ble.device.lost] BLE device E3:90:77:4B:3F:ED (Apple, Inc.) lost.
192.168.1.0/24 > 192.168.1.104 » [15:47:56] [ble.device.lost] BLE device D0:D0:03:21:41:B5 (Samsung Electronics Co., Ltd.) lost.
192.168.1.0/24 > 192.168.1.104 » [15:47:56] [ble.device.lost] BLE device 28:39:5E:41:4F:66 (Samsung Electronics Co., Ltd.) lost.
192.168.1.0/24 > 192.168.1.104 » [15:47:56] [ble.device.lost] BLE device BRC1H 6C:39:75 94:0D:2D:6C:39:75 (Universal Scientific Industrial Co., Ltd.) lost.
192.168.1.0/24 > 192.168.1.104 » [15:47:56] [ble.device.lost] BLE device [TV] Samsung 8 Series (43) 44:5C:E9:B6:C5:44 (Samsung Electronics Co., Ltd.) lost.
192.168.1.0/24 > 192.168.1.104 » [15:47:56] [ble.device.lost] BLE device 6A:4C:FB:CC:A5:9E (Apple, Inc.) lost.
192.168.1.0/24 > 192.168.1.104 » [15:47:56] [ble.device.new] new BLE device detected as 28:39:5E:41:4F:66 (Samsung Electronics Co., Ltd.)
192.168.1.0/24 > 192.168.1.104 » [15:47:56] [ble.device.new] new BLE device BRC1H 6C:39:75 detected as 94:0D:2D:6C:39:75 (Universal Scientific Industrial Co., Ltd.)
192.168.1.0/24 > 192.168.1.104 » [15:47:56] [ble.device.new] new BLE device GBK_H618F_6B65 detected as 60:74:F4:2E:6B:65 (Garmin Ltd.)
192.168.1.0/24 > 192.168.1.104 » [15:47:56] [ble.device.new] new BLE device detected as D0:D0:03:21:41:B5 (Samsung Electronics Co., Ltd.)
192.168.1.0/24 > 192.168.1.104 » [15:47:56] [ble.device.new] new BLE device detected as 48:73:71:81:E4:84 (Apple, Inc.)
192.168.1.0/24 > 192.168.1.104 » [15:47:56] [ble.device.new] new BLE device detected as 49:79:B5:F3:2C:41 (Apple, Inc.)
192.168.1.0/24 > 192.168.1.104 » [15:47:56] [ble.device.new] new BLE device detected as 6A:4C:FB:CC:A5:9E (Apple, Inc.)
192.168.1.0/24 > 192.168.1.104 » [15:47:56] [ble.device.new] new BLE device Arduino_BLE detected as 84:CC:A8:2B:E8:DE (Arduino)
192.168.1.0/24 > 192.168.1.104 » [15:47:56] [ble.device.new] new BLE device detected as 49:73:7B:7E:C8:87 (Apple, Inc.)
192.168.1.0/24 > 192.168.1.104 » [15:47:56] [ble.device.new] new BLE device detected as EE:26:5A:53:40:71 (Apple, Inc.)
192.168.1.0/24 > 192.168.1.104 » [15:47:56] [ble.device.new] new BLE device [TV] Samsung 8 Series (43) detected as 44:5C:E9:B6:C5:44 (Samsung Electronics Co., Ltd.)
192.168.1.0/24 > 192.168.1.104 » [15:47:56] [ble.device.new] new BLE device detected as E2:A6:3D:A5:32:F4 (Apple, Inc.)
192.168.1.0/24 > 192.168.1.104 » [15:47:59] [ble.device.new] new BLE device detected as F2:44:3A:57:12:95 (Apple, Inc.)
192.168.1.0/24 > 192.168.1.104 » [15:48:01] [ble.device.new] new BLE device detected as E3:90:77:4B:3F:ED (Apple, Inc.)
192.168.1.0/24 > 192.168.1.104 » ble.enum b8:27:eb:53:ce:ab^C
Are you sure you want to quit this session? y/n n
192.168.1.0/24 > 192.168.1.104 » ble.enum B8:27:EB:53:CE:AB
192.168.1.0/24 > 192.168.1.104 » [15:48:29] [sys.log] [err] BLE device with address b8:27:eb:53:ce:ab not found.
192.168.1.0/24 > 192.168.1.104 » ble.enum B8:27:EB:53:CE:AB[15:48:53] [ble.device.new] new BLE device detected as b8:27:eb:53:ce:ab
192.168.1.0/24 > 192.168.1.104 » ble.enum 84:CC:A8:2B:E8:DE
[15:48:59] [sys.log] [inf] ble.recon connecting to 84:cc:a8:2b:e8:de ...
192.168.1.0/24 > 192.168.1.104 »

```

Handles	Service > Characteristics	Properties	Data
0001 → 0005	Generic Access (1800)		
0003	Device Name (2a00)	READ	Arduino
0005	Appearance (2a01)	READ	Unknown
0006 → 0009	Generic Attribute (1801)		
0008	Service Changed (2a05)	INDICATE	
000a → 000d	19b10000e8f2537e4f6cd104768a1214		
000c	19b10001e8f2537e4f6cd104768a1214	WRITE, NOTIFY	

```

192.168.1.0/24 > 192.168.1.104 » [15:49:21] [ble.device.lost] BLE device EE:26:5A:53:40:71 (Apple, Inc.) lost.
192.168.1.0/24 > 192.168.1.104 » [15:49:22] [ble.device.new] new BLE device detected as 5B:0E:13:16:13:62 (Apple, Inc.)
192.168.1.0/24 > 192.168.1.104 » [15:49:24] [ble.device.new] new BLE device detected as DF:DF:1A:B3:A0:A8 (Apple, Inc.)
192.168.1.0/24 > 192.168.1.104 »

```

Figure 10: Attaining Arduino information using bettercap

Let's assume the following for illustration purposes:

- MAC Address of Govee Light: 60:74:f4:2e:6b:65
- Handle for Power Control: 0013
- Command to Turn On: 01
- Command to Turn Off: 00

To turn the light on:

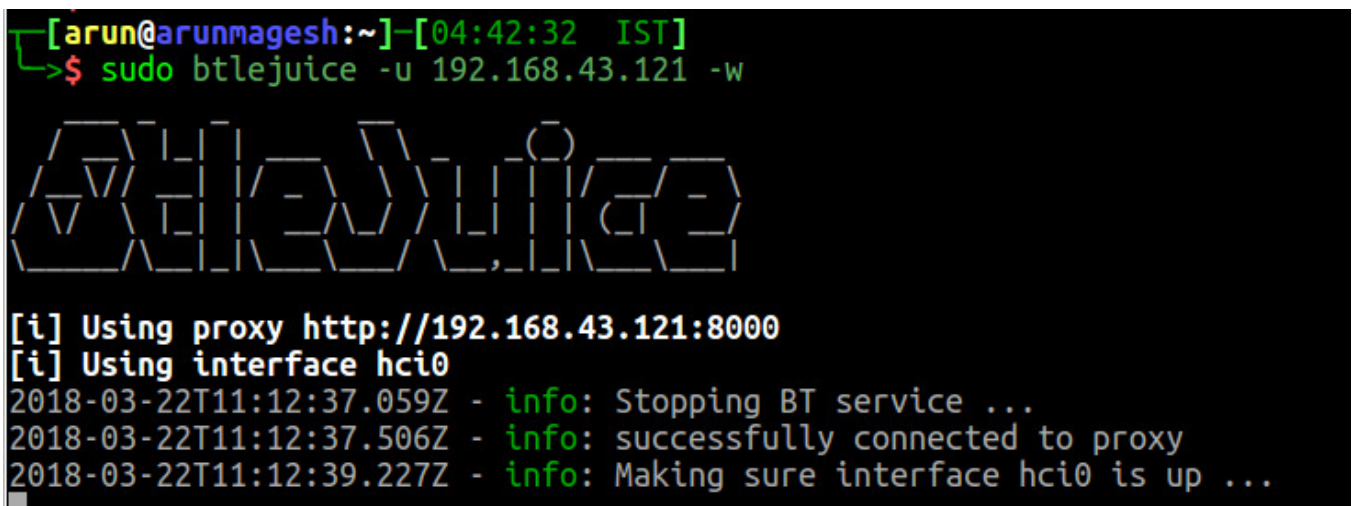
```
ble.write 60:74:f4:2e:6b:65 0013 01
```

To turn it off:

```
ble.write 60:74:f4:2e:6b:65 0013 00
```

BtleJuice:

In our project, we attempted to use the BtleJuice Framework for a Man-in-the-Middle attack on BLE devices. Initially, we faced challenges in setting up BtleJuice, primarily due to having only one Bluetooth adapter, which was insufficient for running both the interception proxy and core components. After adding an additional dongle, we successfully set up BtleJuice. However, during our tests, we encountered limitations in our target device compatibility. While we could detect the Arduino in the device list, attempts to connect resulted in crashes of the Arduino board. The Raspberry Pi, on the other hand, was not detected. This experience highlighted the complexities of conducting BLE MITM attacks and the importance of hardware compatibility in such experiments.

A terminal window with a black background and green and white text. The prompt is [arun@arunmagesh:~]-[04:42:32 IST]. The command sudo btlejuice -u 192.168.43.121 -w is entered. Below the command is a large ASCII art logo for 'BtleJuice'. The output shows status messages: [i] Using proxy http://192.168.43.121:8000, [i] Using interface hci0, and three log entries from 2018-03-22T11:12:37.059Z to 2018-03-22T11:12:39.227Z indicating the stopping of BT service, successful connection to proxy, and ensuring interface hci0 is up.

```
[arun@arunmagesh:~]-[04:42:32 IST]
->$ sudo btlejuice -u 192.168.43.121 -w

BtleJuice

[i] Using proxy http://192.168.43.121:8000
[i] Using interface hci0
2018-03-22T11:12:37.059Z - info: Stopping BT service ...
2018-03-22T11:12:37.506Z - info: successfully connected to proxy
2018-03-22T11:12:39.227Z - info: Making sure interface hci0 is up ...
```

Figure 13: BtleJuice Proxy

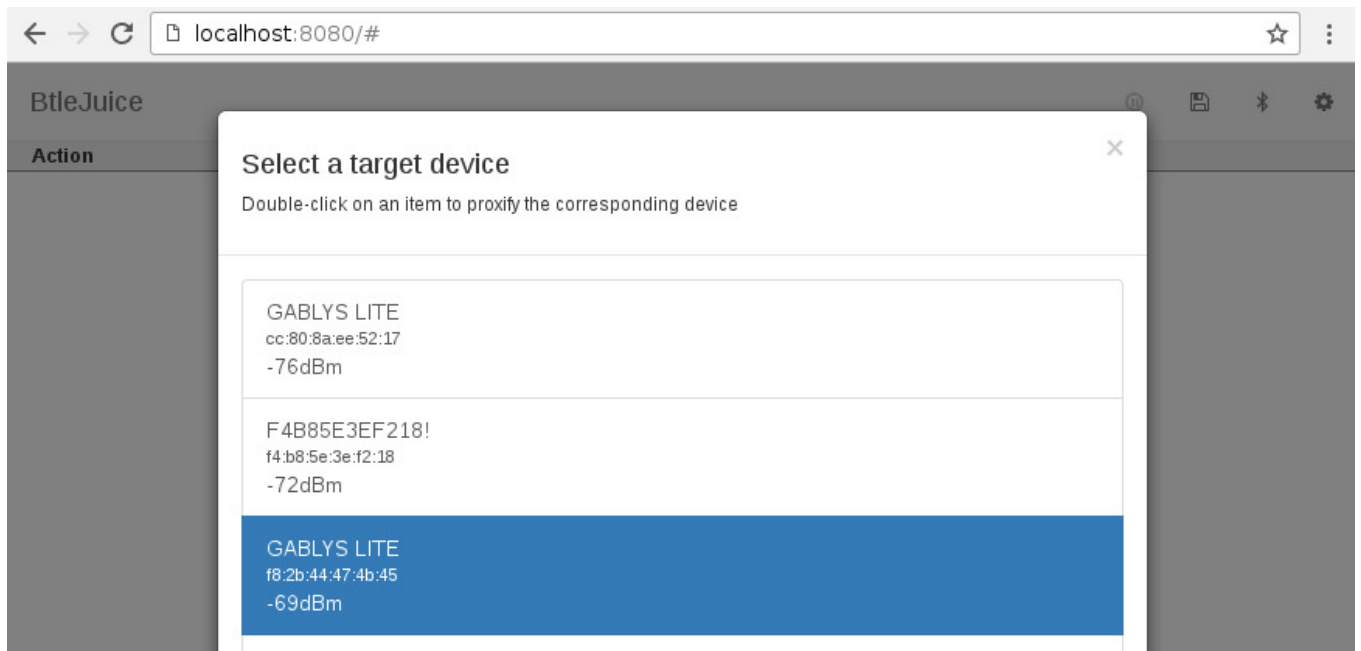


Figure 14: Scanning for BLE device using BtleJuice

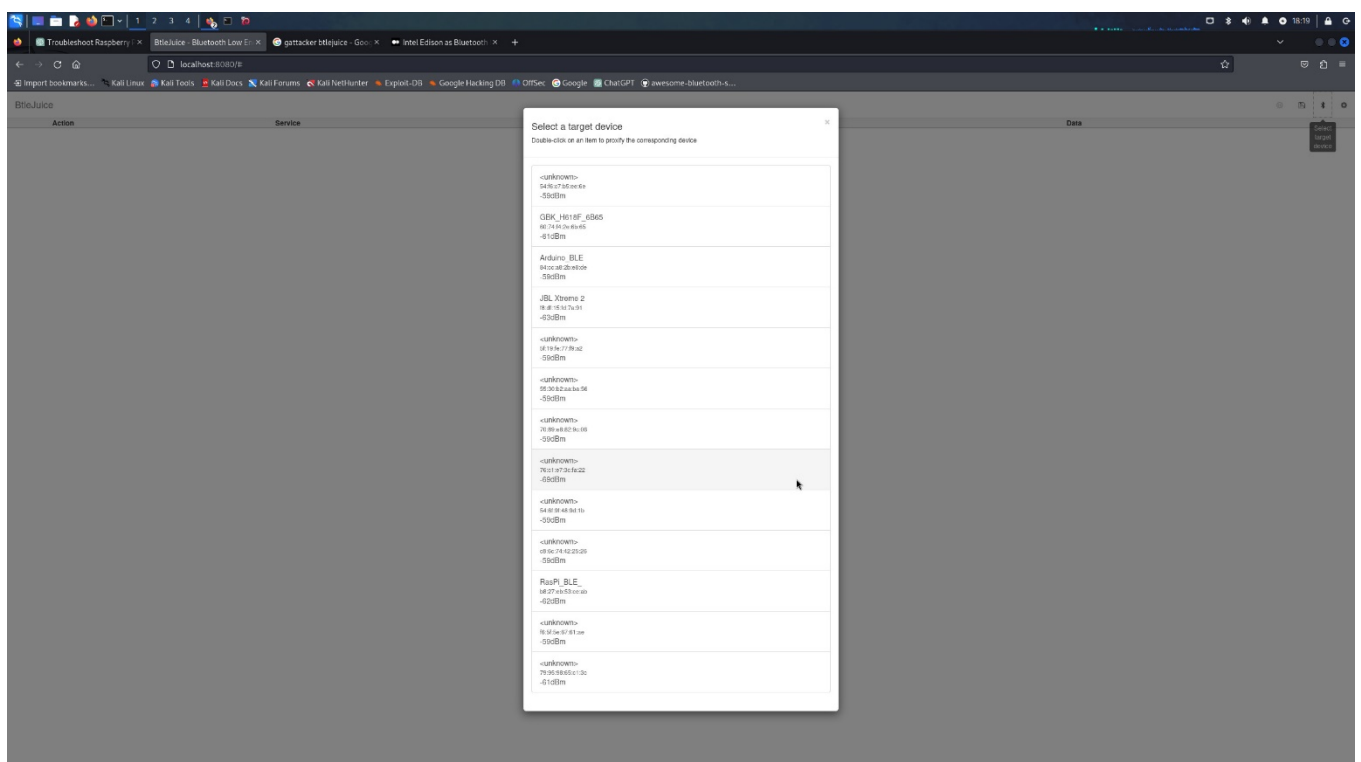


Figure 15: Targeting Arduino using BtleJuice

Blue Deauth:

In our project, we considered the 'Blue Deauth' attack, which encompasses two types: a Ping flood (l2ping) and a Connect flood (rfcomm). This method involves sending a large number of packets to a target device, potentially leading to a Denial-of-Service (DoS) condition. However, we decided against implementing this attack due to concerns about potentially damaging the Arduino hardware. The intensity of the packet flood in both methods could overwhelm the device, causing it to crash or behave unpredictably, which highlighted the need for caution when selecting attack methodologies for our hardware setup.

Conclusion

In conclusion, our study into BLE security provided comprehensive insights into the vulnerabilities and defense mechanisms of this widely-used technology. Through a series of targeted experiments using various tools and attack methodologies, we demonstrated both the resilience and susceptibilities of BLE communications. This project not only highlighted the importance of ongoing security assessments in the rapidly evolving field of wireless communication but also the need for responsible and ethical application of penetration testing tools to safeguard against potential security breaches. Our findings underscore the dynamic nature of cybersecurity and the continuous effort required to protect and enhance the security of BLE-enabled devices.

References

- A survey on Bluetooth Low Energy security and privacy.
- A study of the Feasibility of Co-located App Attacks against BLE and a Large-Scale Analysis of the Current Application-Layer Security Landscape.
- BlueShield: Detecting Spoofing Attacks in Bluetooth Low Energy Networks.
- BLESa: Spoofing Attacks against Reconnections in Bluetooth Low Energy.
- Bluetooth Low Energy Protocol, Security & Attacks.
- A Systematic Review of Bluetooth Security Threats, Attacks & Analysis.
- BLE Injection-Free Attack: A Novel Attack on Bluetooth Low Energy Devices.
- Managing Cybersecurity Break-ins Using Bluetooth Low Energy Devices to Verify Attackers: A Practical Study.
- Key Negotiation Downgrade Attacks on Bluetooth and Bluetooth Low Energy.
- Finding Traceability Attacks in the Bluetooth Low Energy Specification and Its Implementations.
- Security and Privacy Threats for Bluetooth Low Energy in IoT and Wearable Devices: A Comprehensive Survey.
- Security Vulnerabilities in Bluetooth Technology as Used in IoT.