Università degli studi di Modena e Reggio Emilia
Dipartimento di Scienze Fisiche, Informatiche e Matematiche

Corso di Laurea in Informatica

Analisi e simulazione di attacchi a reti BLE insicure

Relatore:
Prof. Luca Ferretti

Candidato:
Salvatore Di Lorenzo

Anno Accademico 2023/24

Indice

Capitolo 1

Introduzione

Ogni sistema informatico viene realizzato sulla base di un trade-off tra varie caratteristiche. Il Bluetooth Low Energy è una tecnologia derivata dal Bluetooth classico e nata per essere impiegata prevalentemente in ambito IoT: i dispositivi target del BLE sono quindi caratterizzati da una potenza di calcolo ridotta ed, eventualmente, da un'autonomia energetica limitata. Mentre il Bluetooth classico mira ad un trade-off bilanciato tra performance e consumi, l'obiettivo del BLE è quello di minimizzare il dispendio energetico sacrificando talvolta le prestazioni e la sicurezza. Nel contesto del BLE, quindi, ogni componente deve essere ottimizzato al fine di utilizzare nel migliore dei modi la poca capacità computazionale a disposizione. Per questo motivo il BLE pone le sue fondamenta su uno stack del tutto indipendente da altre tecnologie e, rispetto al Bluetooth classico, impiega molti accorgimenti per permettere al sistema di funzionare al minor costo possibile. Inoltre, lo stack del BLE presenta un'architettura modulare: in un ambito caratterizzato dall'enorme eterogeneità tra dispositivi, come nell'IoT, questa caratteristica facilita l'integrazione del BLE su diverse categorie di host (ad es.: smartwatch, sensori per l'industria, accessori per smart home, ecc. . .).

Questa tesi vuole approfondire il BLE dal punto di vista della sicurezza delle comunicazioni. In particolare, si tentano di comprendere i principali meccanismi di sicurezza adottati e le eventuali vulnerabilità presenti. In una tecnologia il cui obiettivo principale è quello di ridurre i consumi pur tenendo le prestazioni sufficientemente elevate, le misure di sicurezza adottate risultano spesso non sufficienti: l'obiettivo della tesi è quello di analizzare alcune delle principali modalità di comunicazione offerte dal BLE, ponendo particolare attenzione alle modalità che non impiegano alcuna misura di sicurezza al fine di ottenere prestazioni migliori e consumi ridotti. Il progetto svolto si divide in due macro-sezioni: la prima, dedicata alla replicazione di attacchi, descrive in modo dettagliato tutti i passaggi necessari per sfruttare le vulnerabilità trovate; la seconda sezione invece è dedicata ad un'analisi sperimentale il cui obiettivo è quello di quantificare la perdita di pacchetti che avviene nella fase iniziale della connessione a causa del meccanismo del frequency hopping. Il progetto di tesi mira a scoraggia-

re la trasmissione di dati sensibili in broadcast seppur in forma codificata, mediante l'implementazione di un apposito beacon e successivo reverse-engineering dei pacchetti osservati in rete (sezione 3.2). Vengono inoltre approfonditi gli attacchi man in the middle e replay per evidenziare la possibilità che un dispositivo malevolo riesca ad intercettare pacchetti inviati tra due host connessi tra di loro (sezione 3.3). La presente tesi vuole quindi incoraggiare i programmatori ad utilizzare esclusivamente metodi di comunicazione con garanzie di sicurezza in caso i dati da trasmettere siano caratterizzati da una certa sensibilità. Essendo il BLE una tecnologia che lascia molto spazio ai programmatori data la molteplicità di modalità di comunicazione offerte, infatti, molto spesso viene sacrificata proprio la sicurezza della connessione per cercare di migliorare le prestazioni.

La tesi è strutturata in modo tale da fornire dapprima una descrizione accurata del sistema del BLE, oltre ad un'analisi teorica dei possibili attacchi(capitolo 2). Successivamente, viene presentata l'implementazione effettiva del software dedicato alla replicazione degli attacchi (capitolo 3). In ultimo, viene descritto il software sviluppato e l'ambiente utilizzato per effettuare i test relativi alla perdita di pacchetti causati dal frequency hopping, con relativi risultati correlati da alcuni grafici (capitolo 4).

Capitolo 2

Conoscenze di base

In questo capitolo si descrive nel dettaglio l'architettura e il funzionamento del sistema del Bluetooth Low Energy, ponendo particolare attenzione ai meccanismi necessari alla comprensione del progetto sviluppato. Vengono inoltre illustrate le principali problematiche di sicurezza diffuse nel campo del BLE. In particolare viene fatta un'analisi prettamente teorica degli attacchi replicati nel progetto di tesi. In ultimo, si descrivono gli strumenti e le tecnologie utilizzate sia per l'implementazione del software dedicato alla replicazione degli attacchi, sia per la creazione dell'ambiente di testing impiegato nell'analisi della perdita di pacchetti in fase di inizializzazione della connessione.

2.1 Bluetooth Low Energy

La versione 4.0 del Bluetooth ha introdotto quella che è a tutti gli effetti una nuova tecnologia fondata a partire dal Bluetooth classico: il BLE (Bluetooth Low Energy). Si tratta di una WPAN (Wireless Personal Area Network) che si pone come obiettivo quello di ottimizzare la comunicazione tra dispositivi cercando di bilanciare prestazioni e consumo di energia (da qui, appunto, la denominazione "Low Energy"). Questa caratteristica del BLE è stata sviluppata modificando appositamente alcuni aspetti del protocollo Bluetooth classico, tra cui:

- minore dimensione dei pacchetti trasmessi;
- comunicazione tra le varie parti meno frequente e che avviene solo al momento del bisogno: un dispositivo BLE è capace di entrare in una modalità di "riposo" se non occorre trasmettere dati;
- minore velocità di trasmissione dei dati;

Tutto ciò fa del BLE il candidato perfetto per tutto quello che riguarda la comunicazione tra dispositivi in ambito IoT. Si pensi, ad esempio, alla comunicazione tra un

semplice cardiofrequenzimetro e uno smartphone che deve ricevere i dati riguardanti il battito cardiaco:

- sicuramente i dati trasmessi possono essere inviati in pacchetti di dimensioni ridotte (il battito cardiaco, in fondo, è un semplice numero intero);
- non è richiesto un alto throughput;
- non è importante che ci sia un'altissima velocità di trasmissione dei dati;

Il BLE, difatti, può essere sfruttato appieno proprio in contesti di questo tipo dato che riesce senza problemi a fornire il servizio desiderato preservando, allo stesso tempo, la durata dell'eventuale batteria che alimenta il dispositivo in questione. Importante sottolineare che il BLE non sostituisce assolutamente il Bluetooth nella sua versione classica in quanto, in contesti in cui è richiesta la trasmissione di un grande quantitativo di dati è sicuramente preferibile l'utilizzo del Bluetooth classico.

2.1.1 Stack BLE

Il Bluetooth Low Energy (BLE) opera su uno stack autonomo e separato rispetto ad altri stack standard, il che contribuisce significativamente a migliorare l'efficienza complessiva del sistema. Questa indipendenza permette al BLE di ottimizzare ogni livello del protocollo per ridurre al minimo il consumo energetico e migliorare le prestazioni. La prima caratteristica che occorre sottolineare relativamente allo stack del BLE è sicuramente la separazione netta dello stack in due sottoparti denominate "Controller" e "Host". In questo contesto, il ruolo più importante è quello svolto dall'Host Controller Interface (HCI), componente che gestisce l'interazione tra host e contoller.

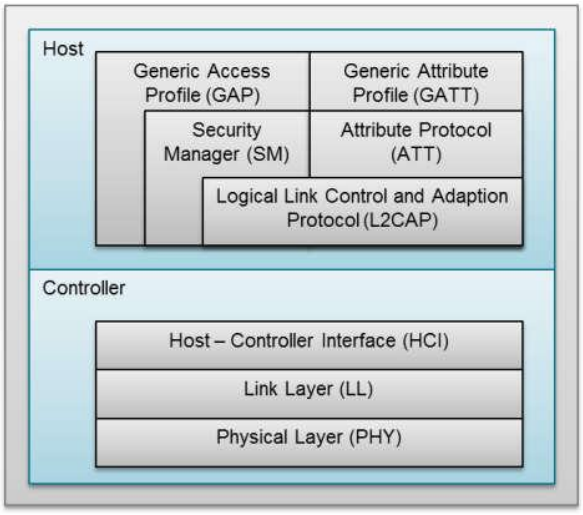

Figura 2.1: Stack BLE

Controller. Il controller opera nella parte più bassa dello stack e si occupa, quindi, di gestire le funzioni radio e tutte quelle operazioni di "basso livello" caratteristiche della comunicazione Bluetooth. In particolare, il controller è caratterizzato dai seguenti livelli (dal basso verso l'alto):

- *Physical Layer*: gestisce tutti gli aspetti legati alla comunicazione radio come, ad esempio, la modulazione del segnale e la gestione delle frequenze di trasmissione.
- *Link Layer*: esegue le funzionalità di "advertising" e scansione e si occupa della procedura di apertura e chiusura di una connessione, riveste un ruolo fondamentale nella scelta del canale da utilizzare tramite l'esecuzione di un determinato algoritmo. Sempre in questo livello viene anche effettuata la gestione degli errori tramite controllo dei codici di ridondanza ciclica (CRC) dei diversi pacchetti.

Host. Nella parte superiore dello stack opera, invece, l'host: in questi strati vengono effettuate tutte quelle operazioni definite di "alto livello" quali la segmentazione del flusso di dati, l'autenticazione, l'associazione e la gestione dei profili e dei servizi. In particolare, l'host è costituito dai seguenti livelli (dal basso verso l'alto):

- *Logical Link Control and Adaptation Protocol (L2CAP)*: svolge il ruolo di un vero e proprio multiplexer consentendo a diverse applicazioni software di operare sulla stessa comunicazione fisica, inoltre si occupa della frammentazione del flusso di dati in modo tale da creare pacchetti della dimensione massima consentita dal Link Layer (MTU).
- *Security Manager Protocol (SMP)*: gestisce tutti gli aspetti relativi alla sicurezza del BLE, in particolare entra in gioco durante la fase di "pairing".
- *Generic Access Profile (GAP)*: stabilisce le modalità di interazione tra i vari dispositivi gestendo, quindi, la fase di discovery, quella di advertising e quella di connessione effettiva. Si occupa, inoltre, della definizione del ruolo del dispositivo nel contesto della comunicazione in corso.
- *Attribute Protocol (ATT)*: gestisce i dati veri e propri da scambiare durante la comunicazione, infatti l'unità di dato più piccola che è possibile trasmettere viene definita proprio come "attributo".
- *Generic Attribute Profile (GATT)*: si tratta del livello più alto dello stack che, di fatto, si interfaccia con l'utente finale. Il suo scopo è quello di raggruppare i vari attributi presenti in strutture dati più complesse denominate "servizi" e fornire procedure per l'accesso a essi.

Host Controller Interface. La suddivisione dello stack in due sottoparti rende l'architettura del BLE totalmente modulare. Questa caratteristica rende di fatto possibile la realizzazione di un dispositivo BLE in cui host e controller siano fisicamente

separati e, soprattutto, appartenenti a produttori distinti. Nella gran parte dei casi, comunque, il ruolo di host viene ricoperto dal sistema operativo del dispositivo in questione, il quale implementa, nel suo hardware, un modulo apposito che riveste il ruolo del controller. Questo tipo di architettura facilita, ovviamente, la portabilità del BLE su diverse piattaforme. Si pensi, ad esempio, al caso di diversi dispositivi IoT in possesso dello stesso controller ma basati su sistemi operativi differenti l'uno dall'altro: è sufficiente scrivere un modulo software apposito per ogni sistema senza preoccuparsi dell'hardware sottostante per permette la comunicazione tra i diversi dispositivi.

La separazione della struttura in due parti distinte implica, però, la presenza di un qualche metodo che possa permettere la comunicazione tra host e controller: questo ruolo chiave viene affidato ad una interfaccia standard (Host Controller Interface). La presenza di questo layer all'interno dello stack è essenziale per rendere modulare a tutti gli effetti l'intera architettura. Questa interfaccia è, difatti, una vera e propria implementazione del design pattern "Hardware Abstraction" nel contesto del BLE: l'unità software (in questo caso identificata dall'host) non dipende direttamente dall'unità hardware (il controller) bensì da un'interfaccia standard, che a sua volta astrae l'hardware sottostante. A partire da questo design, potenzialmente, sarebbe possibile intercambiare diversi host e diversi controller e preservare comunque la funzionalità del sistema, in quanto l'unica dipendenza dei componenti è rappresentata dall'interfaccia standard. Più precisamente, il layer definito come Host Controller Interface (HCI) definisce un'interfaccia standard che abilita la comunicazione tra host e controller. In particolare l'host può inviare "comandi" al controller, che a sua volta può avvertire l'host successivamente all'avvenimento di certi "eventi". Si può immaginare un comando come una richiesta a una certa funzione API e un evento come il valore di ritorno della funzione stessa. Difatti, durante il reale funzionamento del BLE, l'host invia comandi al controller e quest'ultimo, una volta completata l'operazione richiesta, invia un messaggio (evento) all'host. Tuttavia, l'aspetto da tenere in considerazione, è che il controller ha la possibilità di inviare eventi all'host corrispondente anche senza aver ricevuto nessun comando: è questo il caso, ad esempio, della richiesta di connessione da parte di un dispositivo esterno. Nel caso illustrato in figura 2.2 l'host A invia alcuni comandi al controller A e in tutti i casi è possibile notare l'evento "Command complete" che notifica all'host la corretta esecuzione del comando inviato. Si può anche notare, però, la presenza di un altro tipo di evento descritto come "LE Advertising Report", il quale non è stato sollecitato dall'host A bensì da un messaggio inviato dal controller B, appartenente ad un altro dispositivo. In breve, host e controller dello stesso dispositivo comunicano tramite l'HCI, mentre la comunicazione tra dispositivi diversi avviene tramite i rispettivi controller che, una volta ricevuto un messaggio, notificano il corrispondente evento al proprio host.

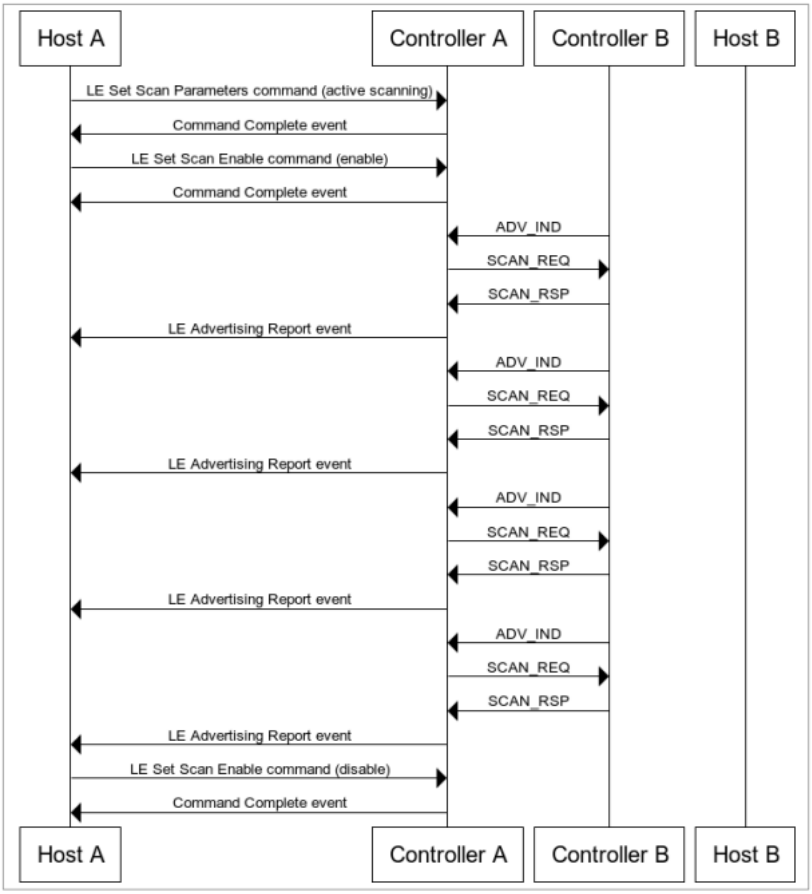

Figura 2.2: Interazione host-controller tra due diversi dispositivi

2.1.2 Livelli superiori dello stack

Ai fini della corretta comprensione del progetto svolto occorre approfondire i livelli più alti dello stack: Generic Access Profile (GAP), Generic Attribute Profile (GATT) e il protocollo su cui si basa, ovvero l'Attribute Protocol (ATT).

Generic Access Profile. Il GAP stabilisce in che modo, in che "vesti", un dispositivo accede ad una rete Bluetooth definendone in primis il ruolo al suo interno e, in secondo luogo, gestendo tutte le fasi della comunicazione stessa. I ruoli definiti e assegnati dal GAP ad un dispositivo in procinto di partecipare ad una comunicazione Bluetooth sono quattro e vengono scelti in base al tipo di operazioni che il dispositivo stesso ha intenzione di svolgere all'interno della rete:

- *Broadcaster*: si tratta di un dispositivo il cui unico scopo è quello di inviare pacchetti "pubblicitari", tecnicamente chiamati "advertise". All'interno dei pacchetti possono essere presenti dati di qualsiasi tipo e possono essere ricevuti da qualsiasi altro dispositivo in ascolto. Una caratteristica fondamentale è rappresentata dall'impossibilità di stabilire una connessione con un dispositivo di questo tipo. Nell'ambito del Bluetooth, un broadcaster viene anche identificato con il nome

"beacon", il quale è un dispositivo che, ad esempio, può essere collocato nei pressi di un quadro in un museo e fornire informazioni relative all'opera ogni qual volta un visitatore si avvicini.

- *Observer*: ruolo del tutto speculare al broadcaster, difatti si tratta di un dispositivo che funge da ascoltatore passivo ("osserva" la rete e non invia nessun pacchetto).
- *Peripheral*: dispositivo concettualmente simile al ruolo svolto da un comune server, rimane infatti in ascolto di eventuali richieste di connessione provenienti da dispositivi "central".
- *Central*: si tratta del ruolo più importante all'interno di una rete dato che ha la possibilità di inizializzare una connessione con un dispositivo "peripheral". Questo ruolo può essere paragonato al ruolo di client caratteristico di una qualsiasi connessione client-server.

Nonostante i ruoli riportati di sopra siano abbastanza rigidi e definiti, è importante sottolineare che non sono fissi per tutta la durata della comunicazione (un dispositivo central potrebbe invertirsi con uno peripheral).

L'altra funzione del GAP è quella di gestire la modalità di comunicazione e/o lo "stato corrente" del dispositivo in questione all'interno della rete. In pratica, le operazioni principali che il GAP offre sono:

- Gestione della modalità di scansione attiva (in questa modalità si richiede attivamente ad un determinato dispositivo l'invio di pacchetti di advertising in modo da ricevere più informazioni su di esso).
- Gestione della modalità "discoverable" (in caso un dispositivo non sia discoverable non risponde a richieste di scansione attiva).
- Gestione della modalità di advertising e impostazione di tutti i relativi parametri come, ad esempio, i dati pubblicizzati e il tempo che deve intercorrere tra un advertise e un altro.
- Inizializzazione di una connessione vera e propria con un altro dispositivo.

Tutte queste operazioni vengono eseguite tramite comandi inviati al controller tramite la HCI e, successivamente, completati dai livelli sottostanti.

Attribute Protocol. L'Attribute Protocol (ATT) definisce la più piccola unità di dato trasmissibile, denominata "attributo", ed è la sua implementazione nello stack che rende possibile il corretto funzionamento del livello immediatamente superiore, il GATT. Più precisamente, un singolo attributo ha una struttura logica ben precisa divisa in quattro parti:

- La handle, corrispondente ad un identificatore univoco necessario per l'accesso all'attributo da parte di un altro dispositivo.
- Il tipo, corrispondente ad un UUID che identica, per l'appunto, che tipo di dato contiene l'attributo in questione.
- Il valore vero e proprio dell'attributo.
- I permessi, che definiscono in che modo un dispositivo esterno può interagire con l'attributo stesso. Nell'implementazione standard dello stack, i permessi relativi ad un attributo vengono stabiliti direttamente dal livello superiore, cioè dal GATT.

Occorre specificare, comunque, che si tratta solo di una suddivisione logica e che, in base all'implementazione del protocollo ATT, potrebbero risultare strutture di dimensione differente.

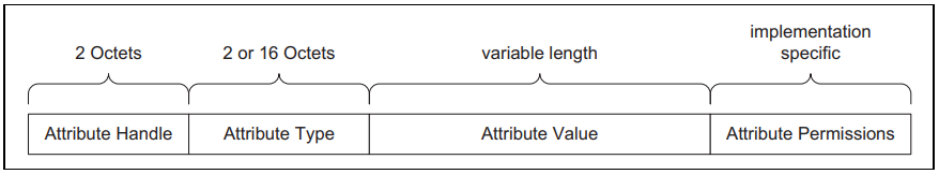

Figura 2.3: Struttura di un attributo

Oltre a definire il concetto di "attributo", il protocollo ATT definisce anche il formato dei pacchetti dedicati proprio allo scambio del valore contenuto negli attributi stessi, memorizzati su diversi dispositivi connessi tramite BLE. Il Protocol Data Unit (PDU) dell'ATT ha una struttura molto semplice, costituita solo da tre parti al massimo:

- Opcode, serve a distinguere le varie operazioni possibili. Grazie a questo campo è possibile distinguere, per esempio, un pacchetto di ATT Request da uno di ATT Response.
- Parametri, sono compresi nel PDU in numero variabile in base all'operazione specificata dall'opcode e servono per dare ulteriori informazioni sull'operazione da svolgere. Un semplice esempio può essere dato dal PDU di un ATT Read Request che richiede al destinatario la lettura di un certo attributo identificato da una precisa handle: in questo caso il parametro dell'operazione corrisponderà alla handle dell'attributo da leggere.
- Firma di autenticazione, permette al mittente di inviare un pacchetto firmato in modo tale da assicurare l'autenticità delle informazioni trasmesse. Si tratta di un campo opzionale e non rilevante ai fini del progetto svolto.

Per chiarire ulteriormente il concetto si pensi all'instruction set di un qualsiasi processore: si ha il nome dell'istruzione stessa, per esempio una "add", abbinata ai corrispondenti operandi necessari per completare l'istruzione in questione. Nel caso specifico dell'ATT, l'instruction set è rappresentato da un elenco di opcode standard definiti dal Bluetooth SIG nella *Bluetooth Core Specification 4.2, vol. 3, parte F, sez. 3.4.8* [2]

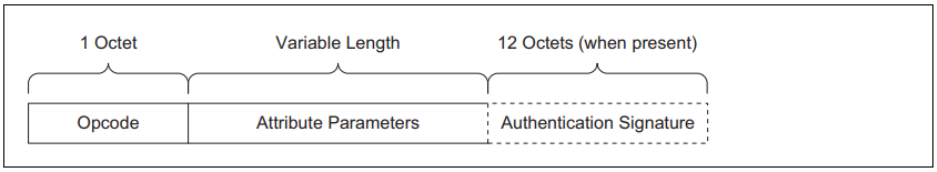

Figura 2.4: PDU del protocollo ATT

Generic Attribute Profile. Altro livello fondamentale per il funzionamento dello stack BLE è, come anticipato, il Generic Attribute Profile (GATT) che, ponendo le sue fondamenta nel layer ATT, si trova in cima allo stack. Il GATT rappresenta l'interfaccia con l'utente finale: in caso, per esempio, si desideri scrivere un'applicazione dedicata all'interazione con un sensore di temperatura occorrerebbe utilizzare le procedure offerte da questo layer sotto forma di API. Questo livello riveste un ruolo di fondamentale importanza siccome definisce il modo in cui i dati sono raggruppati ed esposti da parte di un dispositivo. Il GATT introduce tre concetti utilizzati per creare le strutture dati finali:

- *Caratteristica*: rappresenta l'unità di dato più piccola scambiabile nel contesto del GATT. Può essere pensata come una vera e propria variabile, infatti ogni caratteristica è costituita da un valore e un tipo, oltre ad alcune proprietà che definiscono in che modo una determinata caratteristica può essere utilizzata dalle procedure offerte dal GATT. Possono, ad esempio, essere presenti caratteristiche che abilitano la lettura ma non la scrittura.

- *Descrittore*: si tratta di un dato associato ad una caratteristica e serve a descriverne alcune ulteriori proprietà come, ad esempio, l'unità di misura del dato contenuto nella caratteristica associata. Una caratteristica può avere zero o più descrittori associati. Un descrittore, quindi, può essere visto come un metadato.

- *Servizio*: unità di dato più grande che, infatti, racchiude al suo interno diverse caratteristiche. Se si abbinano le caratteristiche a delle variabili, un servizio può essere paragonato ad una struttura. Esattamente come nelle strutture, infatti, è possibile leggere o scrivere singolarmente le varie caratteristiche in base, ovviamente, alle proprietà associate a ciascuna di esse.

Come anticipato nella sezione 2.1.2, il GATT si basa completamente sul protocollo ATT: proprio per questo motivo ognuno dei tre concetti descritti sopra corrisponde,

in realtà, ad un singolo attributo. Si può, quindi, affermare che l'ATT offre gli "strumenti" per la costruzione di dati complessi ma è il GATT che si occupa di costruirli e "impacchettarli" al fine di esporli alla rete Bluetooth.

Un'altra precisazione importante è che ognuno dei tre oggetti definiti dal GATT ha un "tipo" che, però, non corrisponde ai comuni tipi di dato (intero, float, double, ecc...). Essendo un protocollo di alto livello, il GATT utilizza dei tipi molto più precisi che hanno lo scopo di identificare esattamente il dato rappresentato da un certo oggetto. Ogni tipo standard è identificato da un UUID a 16 bit, stabilito dal Bluetooth SIG e reperibile nel documento ufficiale degli *Assigned Numbers, cap. 3* [4]. Vi è poi la possibilità di utilizzare anche tipi personalizzati per implementare funzionalità non standardizzate: in questo caso al programmatore viene offerta la possibilità di identificare gli oggetti con UUID a 128 bit. Nella figura d'esempio 2.5, un servizio è

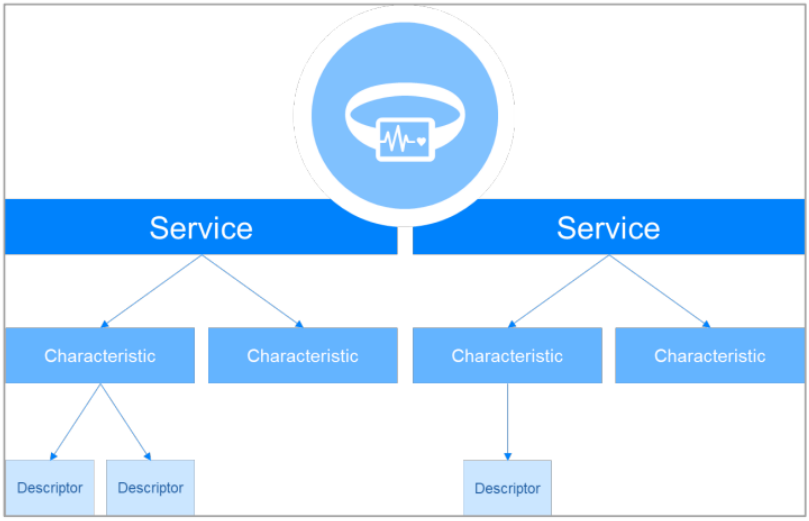

Figura 2.5: Possibile configurazione di un profilo GATT di un cardiofrequenzimetro

sicuramente dedicato alla misurazione del battito cardiaco e raggruppa due caratteristiche: il valore del battito cardiaco all'ultima misurazione e il tempo trascorso dall'ultima misurazione. Un secondo servizio può essere, invece, dedicato alla configurazione del dispositivo stesso come, ad esempio, un tempo standard che deve intercorrere tra una misurazione e l'altra. Come è possibile notare, solo alcune delle caratteristiche presenti sono associate a uno (o più) descrittori.

Oltre a definire le strutture dedicate alla memorizzazione dei dati da esporre alla rete Bluetooth, il GATT stabilisce anche le procedure con cui vari dispositivi possono interagire l'uno con l'altro. Innanzitutto, il GATT è un layer che basa il proprio funzionamento sull'architettura client - server. In particolare, viene identificato come server il dispositivo che contiene i dati e come client il dispositivo che vuole aver accesso a quei determinati dati. Una considerazione importante è che, nell'ambito di una comunicazione GATT, non è sempre vero che il dispositivo client riveste il ruolo di central e il dispositivo server corrisponde al ruolo peripheral. Per quanto sia così nella maggior

parte dei casi, infatti, i ruoli definiti dal GAP sono totalmente indipendenti dal concetto di client-server utilizzato, invece, dal GATT. Ritornando all'esempio precedente, il cardiofrequenzimetro da polso costituisce il server in quanto colleziona dati relativi al battito cardiaco mentre il client potrebbe essere rappresentato da uno smartphone che, connettendosi al cardiofrequenzimetro, permette di visualizzare, ad esempio, lo storico delle misurazioni. Per quello che riguarda la comunicazione vera e propria tra due dispositivi, è previsto l'impiego di pacchetti strutturati secondo le specifiche dell'ATT. Il GATT, in particolare, non fa altro che "compilare" i due campi presenti all'interno del PDU definito dall'ATT (opcode ed eventuali parametri) e descritti nella sezione 2.1.2. Occorre tenere a mente, infatti, che ogni dato offerto da un server GATT non è altro che un attributo sia che si tratti, per esempio, del tipo di una caratteristica sia che si tratti del valore di un descrittore e, proprio per questo motivo, l'unico modo per interagire con i dati presenti su un server risulta essere quello definito dal protocollo ATT. Tra i principali tipi di PDU si hanno:

- *Command*, viene inviato dal client e non richiede alcuna conferma di ricezione da parte del server.

Figura 2.6: Interazione tra client e server nel caso di un Command

- *Request*, viene inviata dal client e il server deve rispondere con una Response entro 30 secondi.

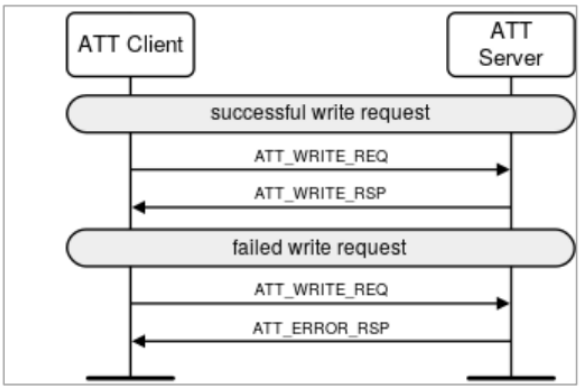

Figura 2.7: Interazione tra client e server nel caso di una Request

- *Notification*, viene inviata dal server e non richiede alcuna conferma di ricezione da parte del client (si contrappone al Command).

Figura 2.8: Interazione tra client e server nel caso di una Notification

- *Indication*, viene inviata dal client e il server deve rispondere con una Confirmation entro 30 secondi (si contrappone alla Request).

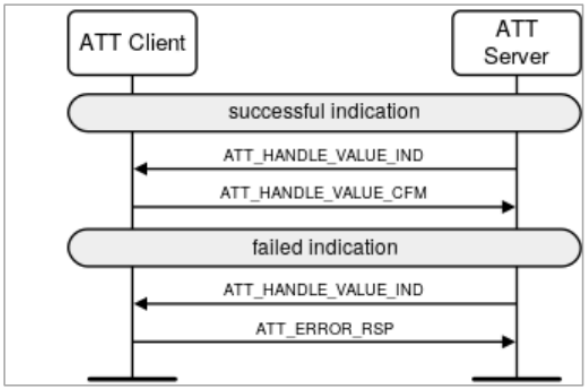

Figura 2.9: Interazione tra client e server nel caso di una Indication

Il tipo di messaggio specifico viene poi identificato, come spiegato prima, proprio dal campo opcode del PDU. Una possibile interazione tra lo smartphone (client) e il cardiofrequenzimetro di esempio (server) potrebbe essere costituita dai seguenti passaggi:

1. Lo smartphone desidera leggere l'ultima misurazione del battito, invia quindi una Request al cardiofrequenzimetro e rimane in attesa di una risposta.

2. Una volta ricevuta la Request, il cardiofrequenzimetro procede ad inviare una Response contenente i dati desiderati.

3. Lo smartphone desidera ora impostare il tempo standard tra una misurazione e l'altra, invia quindi un Command al cardiofrequenzimetro. Il client, in questo caso, non si aspetta di ricevere alcuna conferma.

2.1.3 Indirizzi Bluetooth

Altra nozione importante ai fini della comprensione del progetto è il modo in cui un dispositivo viene identificato all'interno della rete Bluetooth. In linea generale, ogni dispositivo viene identificato da un indirizzo con caratteristiche del tutto simili ad un comune indirizzo MAC utilizzato dalle schede di rete. La differenza sostanziale relativa

al Bluetooth, però, è data dalla presenza di diverse categorie di indirizzi e dalla possibile coesistenza di più indirizzi sullo stesso dispositivo. In particolare, si distinguono due categorie di indirizzi, entrambe caratterizzate da una dimensione di 48 bit, esattamente come nel caso di un classico MAC: public address e random address.

Public Address. Si tratta dell'indirizzo hardware assegnato dal produttore al controller Bluetooth del dispositivo in questione, esattamente come un indirizzo MAC per una scheda di rete. Un indirizzo di questo tipo, essendo parte dell'hardware, rimane fisso e non può essere modificato in alcun modo. Viene definito indirizzo pubblico dato che permette di identificare univocamente un dispositivo all'interno di una rete.

Random Address. Si tratta di un indirizzo generato casualmente via software e assegnabile al proprio dispositivo in modo tale da "mascherare" l'indirizzo hardware (pubblico). A questa categoria di indirizzi appartengono, più nello specifico, due sottocategorie differenti:

- *Random static address*, il quale si distingue dall'indirizzo hardware per via del fatto che deve rimanere fisso per tutta la durata di accensione del dispositivo ma che, eventualmente, è possibile modificare ad ogni riavvio.
- *Private device address*, il quale rappresenta il modo migliore per "mascherare" l'indirizzo hardware di un dispositivo ed evitare eventuali fattori di rischio verso la privacy. Questa categoria di indirizzi, infatti, permette la generazione di un indirizzo casuale anche durante lo stesso power cycle.

Occorre specificare, comunque, che un dispositivo rimane riconoscibile da un altro dispositivo precedentemente associato nonostante l'utilizzo di indirizzi casuali. Difatti esistono altri fattori per verificare l'autenticità della connessione tra cui, il più importante, lo scambio di chiavi crittografiche dette Long Term Key (LTK). Grazie a questo meccanismo un dispositivo può autenticarsi e "farsi riconoscere" da un dispositivo associato anche in caso dovesse avere un indirizzo diverso rispetto a quello avuto durante la prima connessione. Un aspetto fondamentale da tenere in considerazione, dal punto di vista della sicurezza, è che un indirizzo modificabile via software permette di mascherare l'indirizzo hardware aumentando di fatto la privacy ma, allo stesso tempo, potrebbe essere modificato a piacimento da un eventuale attaccante in modo tale da impersonificare un altro dispositivo.

2.1.4 Inizio della comunicazione: fase di advertising

Prima di passare alla fase di comunicazione vera e propria occorre introdurre quello che è un meccanismo fondamentale per il funzionamento della rete Bluetooth: l'advertising (o pubblicizzazione). Si tratta di un concetto, in realtà, molto semplice: se un dispositivo può fare da server GATT o, in altre parole, se ha un profilo GATT con

uno o più servizi da esporre alla rete, allora può "pubblicizzarsi" inoltrando in broadcast pacchetti di advertising contenenti diverse informazioni in modo tale da essere individuato da eventuali client che, se interessati, possono interagire con esso.

PDU di advertising. Il formato dei pacchetti di advertise è definito dal Link Layer dello stack ed è chiaramente costituito da un header e da un payload, contenente i dati che il dispositivo sta effettivamente pubblicando in rete. L'header è costituito da sei

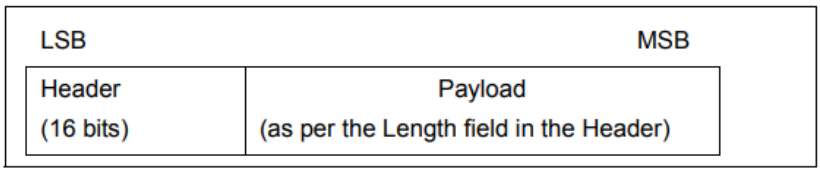

Figura 2.10: PDU di advertising

diversi campi, di cui solo quattro rilevanti e due definiti come Reserved for Future Use (RFU):

- *PDU type*: identifica il tipo di PDU che contiene il pacchetto. Questo campo è utile perché sono presenti diversi tipi di advertising. Per esempio, è possibile che un pacchetto di advertising sia indirizzato ad un dispositivo specifico piuttosto che all'intera rete. Ai fini del progetto svolto, comunque, risulta rilevante solo l'advertising in broadcast, che prende il nome di "connectable undirected advertising" (ADV_IND), dove per "connectable" si intende che il dispositivo mittente del pacchetto accetta effettivamente richieste di connessione da altri dispositivi. Un altro scopo di questo campo, tra l'altro, è proprio quello di identificare il pacchetto dedicato a richiedere l'apertura di una nuova connessione (CONNECT_REQ).

- *Transmit Address (TxAdd)* e *Receive Address (RxAdd)*: forniscono informazioni relative al tipo di indirizzo utilizzato dal mittente (TxAdd) e dal destinatario (RxAdd). In particolare questi campi permettono di distinguere l'utilizzo di indirizzi pubblici e indirizzi casuali. Inoltre non risultano essere sempre significativi in quanto, nel caso di un pacchetto ADV_IND, non è presente nessun destinatario specifico e, di conseguenza, il campo RxAdd non ha significato.

- *Lenght*: indica la dimensione (in byte) del payload in un range da 6 a 37 byte.

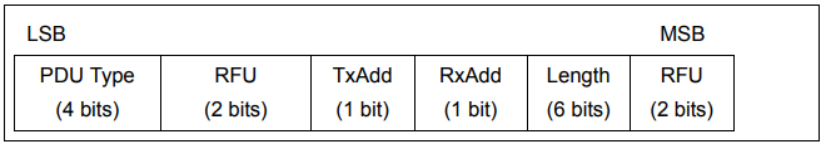

Figura 2.11: Header definito dal Link Layer e utilizzato per l'advertising

Il formato del payload varia a seconda del tipo di PDU specificato dall'header. Ai fini della corretta comprensione del progetto, si approfondisce di seguito il formato del PDU di advertising in broadcast (ADV_IND). La figura 2.12 mostra i due campi

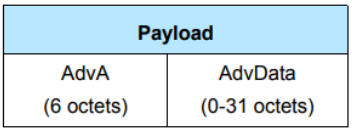

Figura 2.12: Formato del payload dei pacchetti di advertising

presenti nel payload: il primo dei quali, AdvA, rappresenta l'indirizzo del dispositivo "advertiser", corrispondente al mittente del pacchetto. Il secondo campo, denominato come AdvData, è invece dedicato ai dati da pubblicare. Si noti che, quindi, il campo "lenght" presente nell'header non corrisponde alla dimensione dei dati inviati. Più nello specifico, dalla figura emerge che un pacchetto di advertising potrebbe anche non contenere alcun dato.

Per quello che riguarda proprio il campo dedicato ai dati da pubblicare, anch'esso ha una struttura ben definita. Si può immaginare questo campo come un insieme di tuple, ognuna delle quali contenente la dimensione del dato, oltre al dato stesso che, a sua volta, è costituito dal tipo del dato e dal dato effettivo. Il suddetto insieme di tuple deve avere una dimensione massima di 31 byte. In caso di dimensione minore il campo viene completato con byte nulli costituendo, di fatto, una parte non significativa. Osservando la figura 2.13, si noti che:

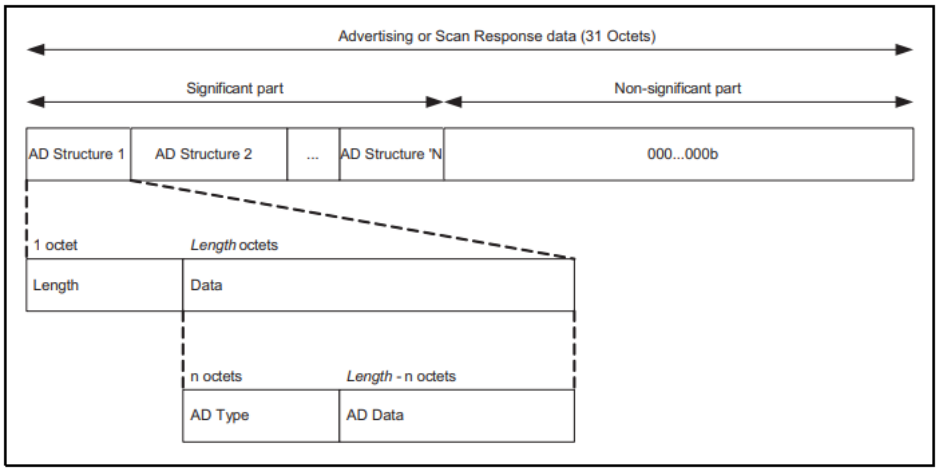

Figura 2.13: Struttura interna del campo AdvData

- Ogni dato da pubblicare ha una propria "struttura".
- Ogni struttura contiene un campo "length" che definisce la dimensione del dato da inviare. Occorre notare che, come emerge dalla figura, questo campo non

definisce la dimensione del dato effettivo, in quanto al suo interno è contenuto anche il tipo del dato stesso.

- La sezione "data" della struttura, invece, è costituita dal tipo del dato (AD Type) e dal dato effettivo. Si noti, come detto prima, che il dato effettivo ha una dimensione pari alla differenza tra la dimensione indicata dal campo "lenght" e la dimensione dell'AD Type.

Il campo Advertising Data Type (AD Type) non ha una dimensione specificata da standard ma tutti i tipi esistenti e visionabili in *Assigned Numbers, sez. 2.3* [4] richiedono l'utilizzo di un singolo byte. Inoltre, il Bluetooth SIG specifica anche i valori validi per ciascun tipo di dato nella Bluetooth Core Specification Supplement, parte A [3].

2.1.5 Diversi tipi di comunicazione

La tecnologia BLE offre diverse modalità di comunicazione, ognuna con vantaggi e svantaggi, che vengono utilizzate in base al contesto di sviluppo dell'applicazione.

Advertising. Il metodo di comunicazione in assoluto più semplice è quello che avviene per mezzo del processo di advertising, descritto nella sezione 2.1.4. Si tratta di una comunicazione non connessa e, il più delle volte, non diretta a nessun destinatario specifico (broadcast). Occorre notare che, nonostante un pacchetto di advertising possa anche essere "directed", ovvero destinato ad un dispositivo specifico, il mezzo trasmissivo rimane comunque broadcast: questo implica la ricezione del pacchetto da tutti i dispositivi in ascolto, pur non essendo i veri e propri destinatari. Si noti, quindi, che l'advertising non offre alcuna sicurezza, in nessuna delle sue varianti: il controller BLE di un dispositivo potrebbe essere configurato in modo da leggere in chiaro anche pacchetti non destinati ad esso in quanto non è presente alcun meccanismo di crittografia. Si utilizza questo metodo di comunicazione, quindi, in casi in cui è richiesta una trasmissione di dati unidirezionale e non è necessario garantire alcuna sicurezza. Un semplice caso d'uso potrebbe essere rappresentato da un dispositivo broadcaster (beacon) presente nei pressi di una determinata opera all'interno di un museo che, inoltrando in broadcast pacchetti di advertising contenenti dati relativi all'opera stessa, permette a tutti i visitatori nei paraggi di ricevere sul proprio smartphone più informazioni a riguardo.

Connessione non sicura. Questo metodo di comunicazione prevede una "connessione" vera e propria tra due dispositivi. Al contrario del processo di advertising, in questo caso la comunicazione è bidirezionale ed è basata sull'architettura client-server. La connessione avviene tra un dispositivo periferico, in ascolto per eventuali richieste di connessione, e un dispositivo centrale che rivestono rispettivamente il ruolo di server e client. Anche per questa modalità non è prevista alcuna misura di sicurezza quindi,

essendo il mezzo trasmissivo broadcast per natura, non è garantita la confidenzialità né l'autenticità dei dati trasmessi.

Oltre alla possibilità di scambio di dati da entrambe le parti, la principale differenza rispetto al semplice advertising è rappresentata dalla possibilità di interagire con strutture dati ben più complesse rispetto a quelle trasmissibili in un pacchetto di advertising. Il "motore" della connessione, infatti, è il livello GATT, grazie al quale un dispositivo periferico può esporre i propri servizi ad un eventuale dispositivo centrale. Un esempio di utilizzo di connessione non sicura può essere rappresentato da un sensore di temperatura: in questo caso è necessaria una comunicazione bidirezionale in quanto il client, ad esempio uno smartphone, deve poter leggere la temperatura dal sensore ma, allo stesso tempo, deve anche avere la possibilità di impostare i parametri del sensore stesso (unità di misura, tempo tra una misurazione e l'altra, notifica in caso di temperatura troppo alta/bassa ...). In questo specifico caso, inoltre, non vi è la trasmissione di dati sensibili e, di conseguenza, non è necessaria la creazione di un canale sicuro.

Pairing. Si tratta del metodo di comunicazione più completo in quanto si basa sul concetto di connessione bidirezionale e, in più, prevede uno scambio di chiavi in modo da creare un canale sicuro tra i due host. Tuttavia la presenza di fasi di cifratura e decifratura dei dati, oltre alla fase di scambio delle chiavi, rende questo metodo il più dispendioso per quello che riguarda le risorse energetiche e, dato che il BLE vede il suo utilizzo soprattutto nel mondo IoT, è utile utilizzare il pairing solo ed esclusivamente in caso sia necessario trasmettere dati sensibili come, ad esempio, il battito cardiaco rilevato da un cardiofrequenzimetro.

2.1.6 Frequency hopping

Per la corretta comprensione di alcuni dati ricavati sperimentalmente durante lo svolgimento del progetto occorre introdurre il meccanismo denominato come "frequency hopping". Il Bluetooth Low Energy è una tecnologia che opera tramite trasmissione radio nella banda dei 2.4 GHz, più precisamente nel range da 2400 MHz a 2483.5 MHz, diviso in 40 canali, ognuno a distanza di 2 MHz. In particolare, 3 canali sono riservati al processo di advertising (37,38,39) mentre gli altri 37 sono dedicati alla trasmissione di dati vera e propria.

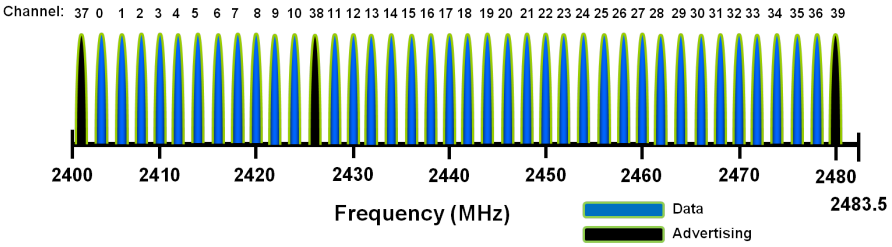

Figura 2.14: Suddivisione della banda in canali

Per quello che riguarda la fase di advertising, un dispositivo seleziona uno o più canali dei 3 riservati e invia i pacchetti sui canali scelti, alternandoli in maniera sequenziale.

Discorso più complesso, invece, per quello che riguarda la selezione dei canali durante il processo di trasmissione dei dati una volta stabilita la connessione: proprio in questo ambito è stato implementato il meccanismo del "frequency hopping" che, per l'appunto, prevede il "salto" tra un canale e l'altro durante la fase di comunicazione tra due dispositivi. Affinché i due host rimangano connessi, però, è necessario che entrambi selezionino lo stesso canale simultaneamente. Per far fronte a questa necessità, durante la fase di inizializzazione della connessione, i due dispositivi coinvolti si "accordano" sul modello dei canali da utilizzare, definito tecnicamente channel map. La channel map non è altro che una maschera di bit che contrassegna i canali disponibili con un bit attivo e quelli non disponibili con un bit a zero. La generazione della channel map viene effettuata dal dispositivo centrale selezionando i canali a partire dai 37 dedicati alla trasmissione dei dati, in base ad alcuni aspetti come, ad esempio, la presenza o meno di interferenze rilevate. Una volta stabilita la channel map da utilizzare, i due host si accordano anche sull'algoritmo dedicato alla selezione effettiva dei canali durante la comunicazione. Si tratta di algoritmi che, a partire dallo stesso canale iniziale e avendo a disposizione la channel map, permettono di operare il frequency hopping e "saltare" tra un canale e l'altro in maniera simultanea.

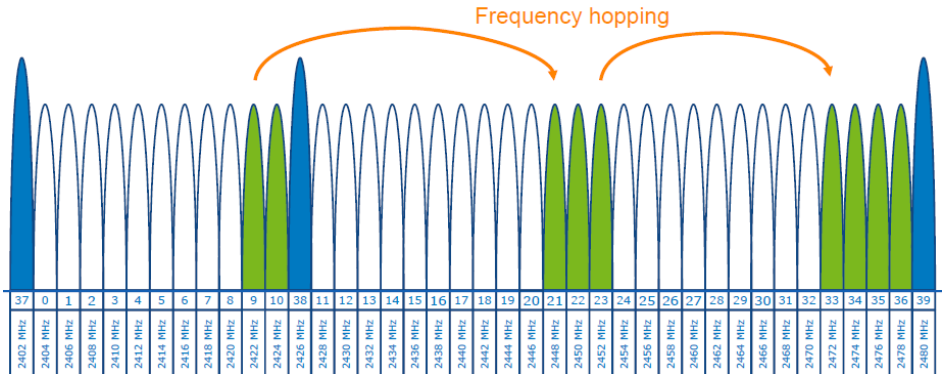

Figura 2.15: Funzionamento del frequency hopping

L'implementazione del frequency hopping, seppur renda la comunicazione tecnicamente più complessa, permette di ridurre notevolmente le interferenze con altri disposi-

tivi radio presenti nelle vicinanze. Un altro effetto, non meno importante, è dato dalla maggiore difficoltà di intercettazione della comunicazione da parte di un attaccante: se la channel map e l'algoritmo utilizzato sono sconosciuti risulta molto complicato tracciare la comunicazione, anche perché il dispositivo centrale potrebbe indicare l'utilizzo di una nuova channel map in qualsiasi momento. Occorre sottolineare, comunque, che nel contesto del BLE il meccanismo del frequency hopping non costituisce in alcun modo una garanzia di sicurezza in quanto, se l'attaccante dovesse riuscire ad intercettare la fase di inizializzazione della connessione, disporrebbe di tutto il necessario per saltare tra un canale e l'altro contemporaneamente ai dispositivi vittima.

2.1.7 Fasi della connessione

Tenendo in considerazione tutti i concetti e le meccaniche introdotte in questo capitolo, si introduce di seguito il processo di connessione tra due dispositivi. Si noti che, ai fini della comprensione del progetto svolto, verrà trattata solo la connessione nella sua versione non sicura, la quale risulta comunque molto simile al processo di pairing. Si può dividere l'intero processo in tre fasi principali: advertising, inizializzazione della connessione e chiusura della connessione. Si tenga in considerazione, come caso d'esempio, un dispositivo client/centrale (per esempio uno smartphone) che ha intenzione di connettersi ad un dispositivo periferico/server (per esempio un sensore di temperatura).

Durante la fase di advertising, come spiegato nella sezione 2.1.4, il dispositivo periferico, ovvero quello che espone un profilo GATT con uno o più servizi, inoltra pacchetti di advertising in rete in modo tale da permettere ad eventuali dispositivi centrali di richiedere la connessione. Nel caso d'esempio, quindi, il sensore di temperatura inoltra in rete un pacchetto di advertising ad intervalli regolari utilizzando, sequenzialmente, uno o più canali tra il 37, il 38 e il 39. Un eventuale dispositivo centrale con intenzione di connettersi ad uno specifico dispositivo periferico entra, in questa fase, in una modalità di "scansione". Nella sua versione più semplice, la scansione passiva, il dispositivo centrale si limita a ricevere tutti i pacchetti di advertising provenienti dalla rete. Discorso leggermente diverso, invece, per la modalità di scansione attiva, che vede il dispositivo centrale inviare uno speciale pacchetto di risposta una volta ricevuto l'advertising da un certo host. Questo pacchetto, denominato "Scan Request" (SCAN_REQ), richiede al dispositivo periferico ulteriori informazioni. Il dispositivo periferico, dal canto suo, invia una "Scan Response" (SCAN_RSP), contenente le informazioni richieste.

Durante la seconda fase, dedicata all'inizializzazione della connessione, il dispositivo centrale interessato alla connessione invia un pacchetto apposito denominato "Connection Request" (CONNECT_REQ) sullo stesso canale da cui è stato ricevuto il pacchetto di advertising proveniente dal dispositivo periferico. Il pacchetto CONNECT_REQ ha una struttura simile a quella degli advertising ed è proprio nel payload che viene inclusa la channel map da utilizzare, insieme ad altri parametri come, ad esempio, l'in-

tervallo di tempo dedicato ad ogni canale. Nel caso d'esempio, quindi, lo smartphone invia un pacchetto di richiesta di connessione inserendo come indirizzo destinatario proprio quello appartenente al sensore di temperatura, precedentemente scoperto grazie al pacchetto di advertising ricevuto. Una volta ricevuta (ed eventualmente accettata) la richiesta di connessione, il dispositivo periferico esegue una sincronizzazione col dispositivo centrale sulla base dei parametri ricevuti nel CONNECT_REQ. Inoltre, la modalità di advertising viene disattivata per quello che riguarda il dispositivo periferico, così come viene disattivata la modalità di scansione per il dispositivo centrale. Entrambi i dispositivi entrano in uno stato di vera e propria connessione: in questa

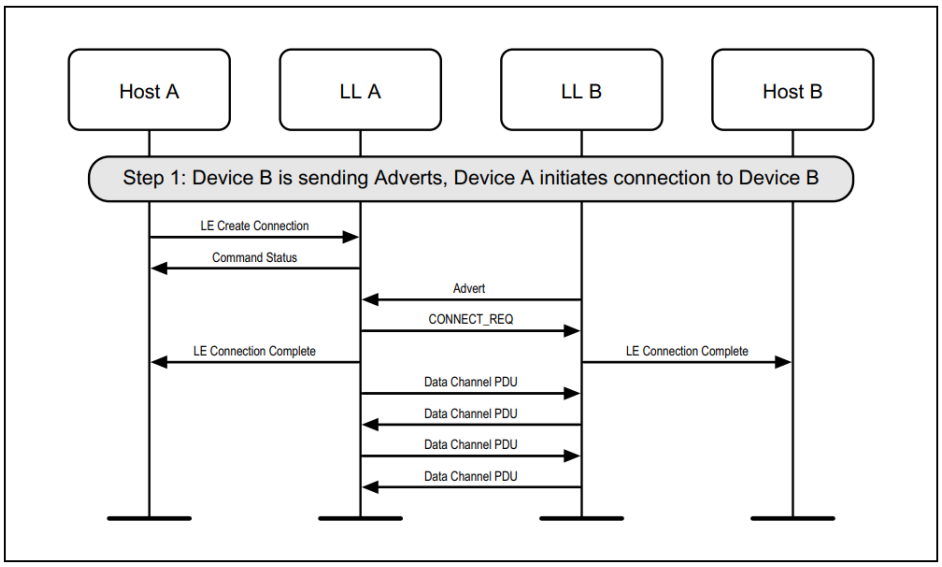

Figura 2.16: Interazione tra Host e Link Layer di dispositivi differenti in fase di inizializzazione della connessione

fase la comunicazione passa dai 3 canali riservati all'advertising agli altri 37 dedicati, invece, alla trasmissione di dati. Entra, quindi, in esecuzione anche l'algoritmo per la selezione dinamica dei canali. La comunicazione in questa fase è gestita dal livello GATT e vengono assegnati i ruoli di client e server.

Nel corso della terza fase viene eseguita la chiusura della connessione che, così come l'inizializzazione, viene gestita dal Link Layer tramite un pacchetto dedicato denominato LL_TERMINATE_IND. Si tratta di un pacchetto di controllo che indica, appunto, la terminazione della connessione con effetto immediato. Il pacchetto riporta, all'interno del payload, un codice di errore che specifica la ragione per cui la connessione sta per essere chiusa. Entrambi i dispositivi connessi possono inviare questo pacchetto. Nel caso d'esempio, una volta completata l'eventuale lettura della temperatura da parte dello smartphone (client), viene inviato al sensore un pacchetto di chiusura della connessione. Il dispositivo periferico, ovvero il sensore, torna infine allo stato iniziale, ricominciando ad inviare pacchetti di advertising.

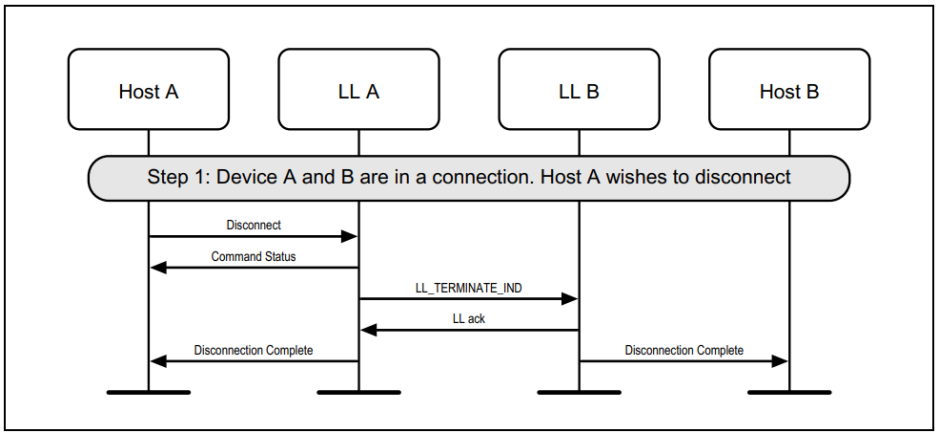

Figura 2.17: Interazione tra Host e Link Layer di dispositivi differenti in fase di disconnessione

2.2 Problematiche di sicurezza

Non essendo un protocollo sicuro di default, il Bluetooth Low Energy viene spesso utilizzato senza pairing in modo da semplificare l'esperienza dell'utente e/o per questioni riguardanti i costi di produzione che, soprattutto in ambito IoT, occorre ridurre dato il grande numero di dispositivi richiesti. Si tenga a mente che il BLE è una tecnologia che, di fatto, nasce per essere un "compromesso" tra prestazioni e consumo di energia: è fondamentale definire la sensibilità dei dati da trasmettere e implementare solo le misure di sicurezza strettamente necessarie. In questa sezione si descrivono proprio le principali problematiche di sicurezza che riguardano il BLE, derivanti sia da un'implementazione errata, sia dalla natura stessa del BLE.

2.2.1 Advertising

Come descritto nel capitolo precedente, il processo di advertising, oltre ad essere parte integrante del processo di connessione, costituisce anche un meccanismo a sé stante per la trasmissione di dati in broadcast. Tenendo in considerazione il formato del payload di advertising mostrato in figura 2.12, uno dei possibili tipi di dato assegnabili è rappresentato dal "Manufacturer Specific Data". Questo particolare tipo di dato viene spesso utilizzato da varie compagnie per definire il proprio formato personalizzato dei pacchetti di advertising. Si tratta, a tutti gli effetti, di una modalità standard utile alla definizione di un'eventuale estensione del protocollo. A differenza di altri tipi standard come, ad esempio, il nome del dispositivo, il controller BLE non è in grado di interpretare i dati trasmessi: occorre, quindi, definire a livello applicativo un metodo utile alla decodifica dei dati ricevuti. La possibilità di utilizzare il tipo "Manufacturer Specific Data" lascia, di fatto, molta libertà al programmatore ed è piuttosto comoda per lo sviluppo di sistemi dedicati ad uno specifico scopo non descritto

da standard. Alcuni esempi degni di nota possono essere rappresentati da iBeacon ed EddyStone, tecnologie proprie rispettivamente di Apple e Google dedicate allo sviluppo di sensori di prossimità. Il problema, in questo caso, nasce proprio dall'elevato grado di libertà lasciato al programmatore, aprendo la strada a possibili implementazioni errate. Avendo la possibilità di trasmettere i dati in un formato non standard, si potrebbe pensare che questo possa rappresentare una misura di sicurezza in quanto, come detto prima, si rende necessario anche un software in grado di decodificare i dati ricevuti: si tratta di un'osservazione del tutto errata. Occorre, prima di tutto, tenere a mente che i pacchetti di advertising vengono trasmessi in broadcast e in chiaro, senza impiegare alcuna tecnica di crittografia: per questo motivo un attaccante potrebbe sicuramente leggere il contenuto del pacchetto tramite uno sniffer senza nessun problema. Una volta ricevuta una quantità discreta di pacchetti, l'attaccante potrebbe utilizzare tecniche di reverse engineering in modo tale da decodificare il contenuto dei pacchetti anche non avendo l'applicativo in grado di farlo legittimamente.

Data Type	Description
«Manufacturer Specific Data» *uint16*, which may be followed by *struct*	The first value contains the Company Identifier Code. Any remainder contains manufacturer specific data.

Figura 2.18: Formato del tipo Manufacturer Specific Data

Si consideri il formato del Manufacturer Specific Data, costituito dai primi due byte contenenti un codice identificativo della compagnia (*Assigned Numbers, cap. 7* [4]) ed eventuali altri byte contenenti la struttura dati vera e propria. Si prenda, come esempio, un semplice cardiofrequenzimetro che, per inviare i dati relativi al battito cardiaco, non richiede la connessione con un dispositivo centrale bensì utilizza un metodo di comunicazione tramite advertising con formato proprio. Un eventuale attaccante, dopo aver collezionato un certo numero di pacchetti, potrebbe riuscire ad identificare i vari campi della struttura personalizzata. Nel caso, piuttosto semplice, in cui il cardiofrequenzimetro trasmetta solo il battito cardiaco, i vari pacchetti ricevuti potrebbero essere costituiti dai primi due byte fissi e da un altro byte variabile: in questo caso, analizzando anche la struttura del Manufacturer Specific Data, si può concludere che i due byte fissi rappresentano l'identificatore della compagnia e, sicuramente, l'ultimo byte contiene le informazioni relative al battito cardiaco. Si tratta di un caso semplificato rispetto alla realtà ma comunque utile a spiegare il problema derivante dalla trasmissione di dati in broadcast tramite advertising, metodo di comunicazione che non offre alcuna misura di sicurezza, indipendentemente dall'utilizzo o meno del tipo Manufacturer Specific Data. Tuttavia, la versione 5.4 del Bluetooth, ha introdotto la funzionalità di "Encrypted Advertising", versione dell'advertising che prevede lo scambio di chiavi crittografiche tra le varie parti della comunicazione. In questa tesi non

si considera però l'uso di tale meccanismo, che ad oggi rappresenta una novità e non sembra essere ancora impiegato in dispositivi commerciali.

2.2.2 Device tracking

Un'altra minaccia alla privacy caratteristica del Bluetooth è quella definita come "Device Tracking". Questo tipo di problematica riguarda, in particolare, i dispositivi "wearable", ovvero tutti quei dispositivi che possono essere indossati dalle persone come, ad esempio, auricolari e smartwatch. Come spiegato nella sezione 2.1.3 , ad ogni controller Bluetooth viene assegnato, dal produttore, un indirizzo univoco detto "Public Address", anche detto indirizzo "hardware" trattandosi di un indirizzo fisso e integrato fisicamente all'interno dell'hardware del controller. Questo aspetto, però, rappresenta un potenziale fattore di rischio in quanto, pubblicando il proprio indirizzo hardware, un dispositivo si rende univocamente riconoscibile all'interno della rete. Questo, sicuramente, non costituisce un problema nel caso il dispositivo in questione sia, ad esempio, un sensore di temperatura posizionato in una stanza o, in generale, un qualsiasi dispositivo non in movimento. Discorso diverso, infatti, per un dispositivo indossato da una persona che, pubblicando il proprio indirizzo hardware, potrebbe permettere ad eventuali osservatori malevoli di tenere traccia di tutti gli spostamenti del proprietario. Questa problematica è, in realtà, di semplice risoluzione tramite l'impiego dei "Random Address": in questo modo anche in presenza di osservatori malevoli, infatti, non è possibile tenere traccia degli spostamenti della persona data la costante variazione dell'indirizzo pubblicato. Questo tipo di indirizzi viene, a tal proposito, utilizzato anche per gli smartphone.

2.2.3 Device spoofing

Con l'introduzione dei "Random Address" si è risolto il problema della privacy ma, trattandosi a tutti gli effetti di un indirizzo definito lato software, si è aperta la strada ad un altro problema altrettanto rischioso per la sicurezza della comunicazione BLE. Tecnicamente, infatti, come descritto dal nome, gli indirizzi di questa categoria dovrebbero essere generati casualmente e autonomamente dal controller BLE con l'unico obiettivo di mascherare l'indirizzo hardware del dispositivo. Esiste, però, un comando per l'HCI utile ad impostare un nuovo indirizzo casuale che, in realtà, non è definito casualmente bensì mediante un parametro abbinato al comando in questione. Il comando "HCI_LE_Set_Random_Address", come specificato nella *Bluetooth Core Specification, vol. 2, parte E, sez. 7.8.4* [2], accetta come parametro un campo di 6 byte corrispondente esattamente all'indirizzo che si desidera impostare. Il problema, in questo caso, è dato dal fatto che questo comando, se utilizzato in maniera malevola, permette tranquillamente di assegnare ad un controller BLE esattamente lo stesso indirizzo di un altro dispositivo: durante la fase di advertising, quindi, il campo

Command	OCF	Command parameters	Return Parameters
HCI_LE_Set_Random_Address	0x0005	Random_Address	Status

Figura 2.19: Specifica del comando per HCI "Set Random Address"

AdvA (figura 2.12) del dispositivo "clone" risulta indistinguibile rispetto all'indirizzo pubblicizzato dal dispositivo "vittima", creando non pochi problemi. Si pensi, infatti, che in presenza di Advertising Address (AdvA) uguali, l'unico fattore che permette la distinzione tra due dispositivi è costituito dai dati che vengono effettivamente inviati nel pacchetto di advertising. Anche in questo caso, però, la clonazione di un dispositivo in fase di advertising risulta assai semplice: si ricorre all'utilizzo di un altro comando per l'HCI grazie al quale è possibile specificare i dati da pubblicizzare come parametro. Il comando "HCI_LE_Set_Advertising_Data", definito nella *Bluetooth Core Specification, vol. 2, parte E, sez. 7.8.7* [2], accetta due parametri: il primo, corrispondente alla dimensione totale della struttura dati trasmessa, serve a specificare il numero di byte significativi del payload (da 0 a 31); il secondo, invece, rappresenta proprio i dati da inviare tramite advertising seguendo la struttura definita nella figura 2.13.

Command	OCF	Command parameters	Return Parameters
HCI_LE_Set_Advertising_Data	0x0008	Advertising_Data_Length, Advertising_Data	Status

Figura 2.20: Specifica del comando per HCI "Set Advertising Data"

Combinando i due comandi presentati in questa sezione è possibile, quindi, replicare esattamente il pacchetto di advertising di un dispositivo. La problematica di sicurezza diventa piuttosto grave in caso il dispositivo clonato rivesta il ruolo di server in attesa di connessioni da un eventuale client: in assenza di meccanismi di autenticazione e, cioè, in caso si tratti di una connessione non basata sul pairing, un dispositivo centrale non riuscirebbe in alcun modo a distinguere il dispositivo periferico legittimo da quello malevolo, aprendo la strada ad attacchi Man In The Middle (MITM) e Replay.

2.2.4 Man In The Middle e Replay Attack

L'attacco conosciuto come "Man In The Middle" costituisce la più grande minaccia alla sicurezza nell'ambito del Bluetooth. Si tratta di un tipo di attacco che colpisce la comunicazione orientata alla connessione, soprattutto per quello che riguarda la sua versione non sicura, e che permette la lettura e la modifica dei dati trasmessi tra un dispositivo centrale e uno periferico. In questo caso, quindi, non si sta parlando di pacchetti di advertising trasmessi in broadcast: questo tipo di attacco ha lo scopo

di interferire con una connessione in atto, superando anche l'ostacolo del frequency hopping.

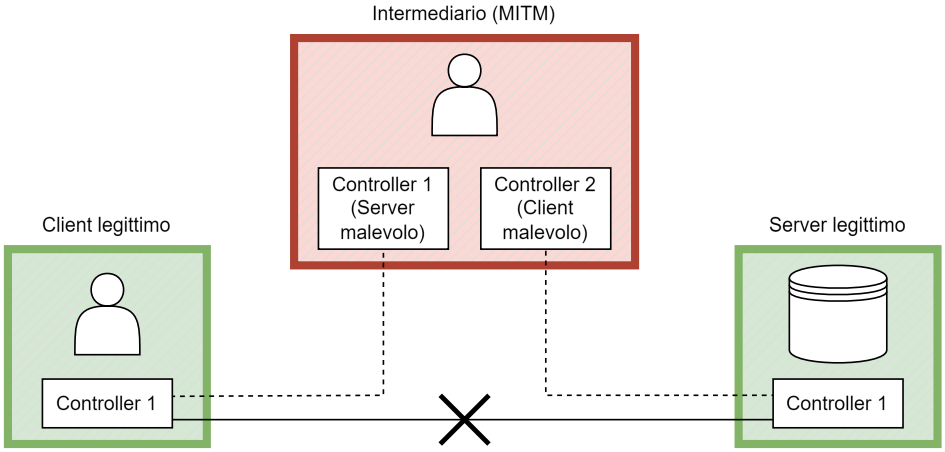

Figura 2.21: Schema architetturale di un attacco MITM eseguito in ambito BLE

Un attacco MITM richiede, appunto, la presenza di un dispositivo che faccia da intermediario tra i due dispositivi vittima e può essere suddiviso in varie fasi:

1. Durante la fase di advertising del server legittimo, l'intermediario riceve e memorizza il pacchetto di advertising inviato da quel server e, combinando i comandi introdotti nella sezione precedente, replica esattamente il pacchetto di advertising del server legittimo. L'obiettivo di questa fase è, quindi, quello di clonare il server legittimo.

2. A questo punto si vogliono massimizzare le probabilità che il client legittimo si connetta al server malevolo: per farlo, il dispositivo intermediario invia una richiesta di connessione al server legittimo che, conseguentemente, termina il processo di advertising. Al termine di questa fase il server legittimo ha stabilito una connessione con il client malevolo e, disattivando la modalità di advertising, il client legittimo non ha più la possibilità di richiedere una connessione. Infine, il server malevolo avvia la fase di advertising utilizzando il pacchetto clonato dal server legittimo.

3. In ultimo, occorre attendere che il client legittimo richieda la connessione al server malevolo che, si ricordi, risulta totalmente indistinguibile dal server legittimo grazie all'impiego dell'address spoofing.

Occorre precisare che l'intermediario, in realtà, necessita di due diversi controller BLE: uno dedicato ad emulare il client legittimo e uno dedicato all'impersonificazione del server legittimo, come illustrato in figura 2.21. Il meccanismo di clonazione appena introdotto viene descritto graficamente di seguito, omettendo alcuni pacchetti non essenziali.

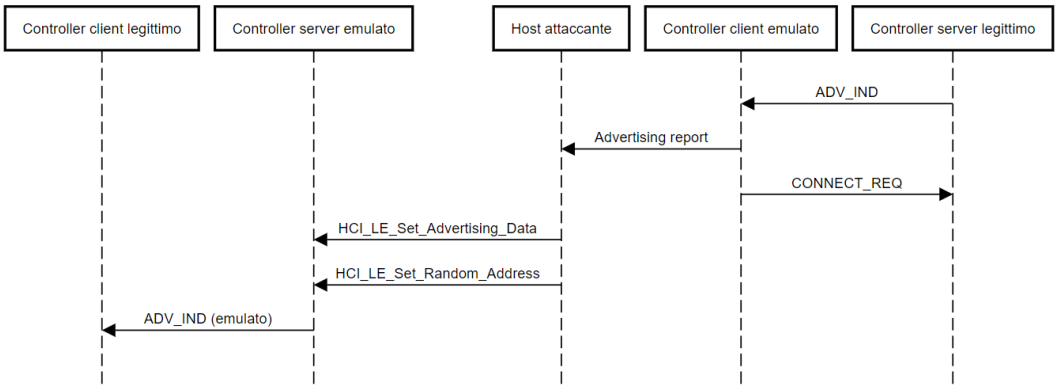

Figura 2.22: Interazione tra le parti nella fase iniziale di un attacco MITM

Una volta avvenuta la connessione del client legittimo al server malevolo, si ha un contesto in cui sono presenti, in realtà, due connessioni distinte. Si tratta di due normali connessioni BLE, ognuna con il proprio distinto modello di frequency hopping ma l'intermediario, possedendo due controller, ha la possibilità di seguire i salti di entrambe le connessioni. Proprio per questo motivo il frequency hopping del BLE non deve essere visto come una misura di sicurezza bensì come un accorgimento tecnico volto a migliorare la qualità della connessione.

La comunicazione vera e propria viene avviata dal client, il quale richiede al server i servizi offerti e altri parametri utili alla sincronizzazione dei due dispositivi vittima. In questa fase l'intermediario si limita ad inoltrare i pacchetti provenienti da entrambe le parti senza modificarne il contenuto, in quanto si tratta di una sorta di "prologo" utile a stabilire la connessione. Successivamente, ogni pacchetto ATT inviato dal client viene ricevuto dal controller assegnato al server malevolo che, sollecitando l'host con un apposito evento, ne comunica la ricezione. A questo punto l'host malevolo può analizzare il contenuto del pacchetto in chiaro seppur non essendo il destinatario legittimo. Una volta analizzato ed eventualmente modificato il pacchetto ricevuto, l'host comunica con il controller assegnato al client malevolo in modo da inviare il pacchetto al server legittimo. Il percorso inverso viene, ovviamente, compiuto da un eventuale pacchetto di risposta ATT proveniente dal server e diretto al client. Di seguito vi è una rappresentazione grafica del processo appena descritto, omettendo però il "prologo" iniziale per semplicità.

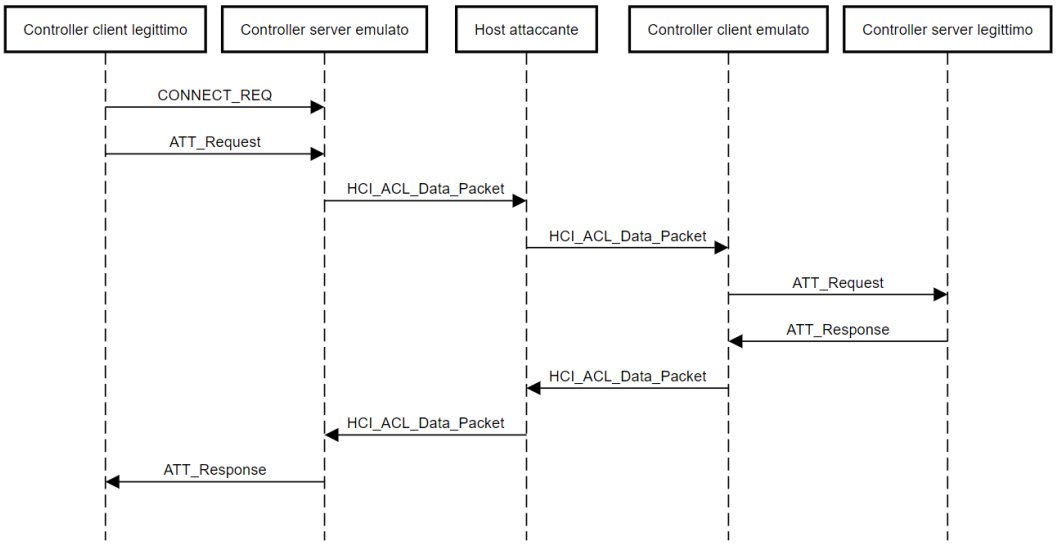

Figura 2.23: Interazione tra le parti in un attacco MITM in corso

Si sottolinea che, a partire da un attacco MITM, può essere effettuato anche un attacco di tipo replay. Quest'ultimo tipo di attacco è costituito dal "re-invio" di pacchetti ad un server legittimo in modo da eseguire operazioni di qualsiasi tipo. Si pensi, per esempio, al caso di un lucchetto che offre le funzionalità di blocco e sblocco previo invio di determinati comandi ATT abbinati ad un codice segreto definito dal client legittimo: tramite un attacco replay si potrebbe inviare il comando di apertura in qualsiasi momento pur non essendo il client legittimo. Chiaramente, in presenza di un canale di comunicazione sicuro, l'unico modo per venire a conoscenza del codice segreto da inviare è quello di porsi come intermediario in una precedente connessione tra client e server legittimi, effettuando un vero e proprio attacco MITM seppur in forma passiva. L'unica differenza rispetto ad un attacco MITM tradizionale, quindi, è data dal fatto che, oltre ad inoltrare i pacchetti provenienti da entrambe le parti, il dispositivo intermediario deve anche memorizzare i pacchetti ricevuti, in modo tale da replicarli al bisogno sotto forma di attacco replay.

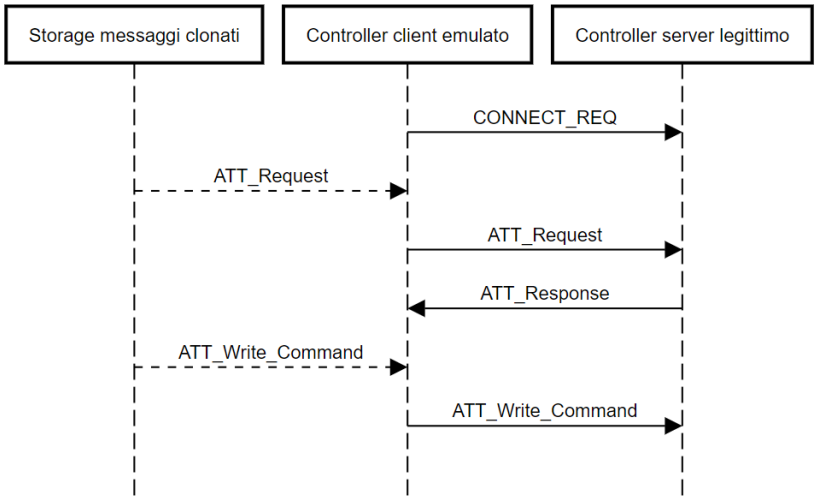

Figura 2.24: Esecuzione di un attacco replay successivamente ad un attacco MITM con memorizzazione dei pacchetti scambiati

Per evitare attacchi di tipo MITM e replay è necessario utilizzare una connessione basata su pairing. Questo metodo di comunicazione, infatti, offre varie garanzie di sicurezza tra cui autenticità e rilevazione di messaggi duplicati. In particolare, in fase di apertura della connessione, è richiesta ad entrambi i dispositivi partecipanti l'esecuzione di una qualche azione di autenticazione (ad esempio, può essere richiesta al client la digitazione di un PIN visualizzabile sullo schermo del server). Al termine della fase di autenticazione, i dispositivi condividono una chiave crittografica che impedisce che i dati trasmessi vengano letti da un eventuale intermediario, il quale non è a conoscenza della chiave necessaria alla decifratura dei dati. Inoltre, al fine di prevenire attacchi di tipo replay, ogni pacchetto viene firmato utilizzando un Message Authentication Code (MAC) generato da un certo algoritmo, in combinazione ad un contatore incrementale: una volta ricevuto un pacchetto firmato con contatore pari ad N, il controller accetta solo ed esclusivamente un pacchetto con contatore pari ad N+1 (*Bluetooth Core Specification, vol. 1, parte A, sez. 5.4.4* [2]). Si noti, però, che non tutti i metodi di pairing offrono una protezione da attacchi MITM e/o replay. In particolare, su dispositivi in cui non sono presenti tastiera e schermo, l'autenticazione tramite PIN risulta impossibile: si utilizza un metodo di pairing denominato "Just Work" caratterizzato dall'utilizzo di una chiave di crittografia generata in fase di connessione e condivisa tra i due dispositivi senza, però, garantire la reale autenticità della comunicazione. Per questo motivo, un eventuale intermediario, potrebbe intercettare la prima connessione tra client e server legittimi e aprire due connessioni separate: l'unica differenza in questo caso è data dal fatto che l'intermediario deve dapprima decifrare e, successivamente, cifrare i pacchetti utilizzando ripetutamente le chiavi corrispondenti al client e server legittimi.

2.3 Tecnologie e strumenti utilizzati

Si introducono, in questa sezione, tutti gli strumenti e le tecnologie utilizzate per lo svolgimento del progetto di tesi.

2.3.1 Espressif ESP32

Si tratta di una gamma di microcontrollori SoC basati su architettura RISC-V. In particolare, sono stati utilizzati i modelli ESP32-C6 ed ESP32-C3, entrambi dotati di Bluetooth 5. Il modo migliore, secondo il produttore, per programmare questo tipo di dispositivi è tramite l'utilizzo di un framework dedicato, denominato Espressif IoT Development Framework (ESP-IDF) [5]. Grazie a questo framework è possibile scrivere programmi integrando anche tutte le funzionalità relative agli eventuali moduli disponibili quali, come detto prima, il Bluetooth o, ancora, il Wi-Fi. Trattandosi di un sistema embedded, occorre un sistema operativo apposito che massimizzi le prestazioni e diminuisca l'utilizzo di memoria ed energia: si utilizza FreeRTOS [1]. Si tratta, in breve, di un kernel di sistema operativo real-time ed open-source che, compilato unitamente al modulo software sviluppato, viene caricato sul microcontrollore e successivamente eseguito. Il linguaggio utilizzato per l'implementazione di programmi eseguibili da FreeRTOS è il C e, caratteristica fondamentale, la programmazione è di tipo "event-driven": ad ogni evento ricevuto come, per esempio, un pacchetto ATT proveniente da un altro dispositivo, viene sollecitata ed eseguita una determinata funzione in risposta all'evento stesso. Questo particolare paradigma di programmazione si contrappone al paradigma procedurale, caratterizzato dall'esecuzione sequenziale del programma sviluppato a partire da una determinata funzione di ingresso.

2.3.2 Nordic Semiconductor nRF52840 Dongle

Si tratta di una chiavetta USB a basso costo prodotta da Nordic con supporto a Bluetooth 5.4, Bluetooth mesh, Thread, Zigbee e altre tecnologie wireless [8]. La chiavetta può essere programmata oppure utilizzata in combinazione ad altri programmi terzi previa installazione di firmware dedicato. Ai fini del progetto svolto, in particolare, non è necessaria la programmazione della chiavetta, in quanto viene utilizzata come interfaccia di cattura esterna, o "extcap", da Wireshark.

2.3.3 Wireshark

Wireshark è un software open-source utilizzato, in ambito network, per scopi di debug, analisi e sviluppo di protocolli. Offre la possibilità di ispezionare, filtrare e ordinare

i pacchetti di rete ricevuti con l'ausilio di un'interfaccia grafica dedicata: può essere, quindi, considerato come un'evoluzione dell'utility da riga di comando "tcpdump". Tramite l'installazione di un apposito modulo software è possibile integrare la chiavetta nRF52840 all'interno dell'ecosistema di Wireshark consentendo, di conseguenza, la ricezione e analisi dei pacchetti Bluetooth in transito. Occorre sottolineare che i pacchetti di advertising sono visionabili anche impostando, come interfaccia di cattura, il controller BLE del dispositivo in questione: il problema, però, è dato dal fatto che un controller BLE "classico" non offre la possibilità di ispezionare pacchetti che non siano diretti al dispositivo host corrispondente. Utilizzando come interfaccia di cattura il controller BLE, quindi, è possibile ispezionare esclusivamente i pacchetti di advertising di tipo "undirected", i pacchetti di advertising diretti al dispositivo in questione e i pacchetti relativi a una connessione avviata tra i cui partecipanti vi è anche il dispositivo stesso. Proprio per ovviare a questo problema si utilizza la chiavetta nRF52840, il cui firmware evita che nessun pacchetto venga scartato a priori, anche se il destinatario legittimo dovesse essere costituito da un dispositivo diverso: si può pensare la suddetta chiavetta come un vero e proprio sniffer (o observer).

Una funzionalità fondamentale offerta da Wireshark è, inoltre, quella di "seguire" automaticamente tutti i salti effettuati da due dispositivi connessi: in questo modo, in assenza di crittografia, è possibile ricevere in chiaro tutti i pacchetti in transito tra i due dispositivi, sebbene la chiavetta non faccia legittimamente parte della connessione. Il funzionamento di questo meccanismo è, in realtà, molto semplice: occorre intercettare il pacchetto di CONNECT_REQ proveniente da un dispositivo; questo tipo di pacchetto, come spiegato nella sezione 2.1.7 contiene, all'interno del payload, tutti i parametri necessari ai dispositivi per l'esecuzione del frequency hopping. Collezionando le informazioni contenute nel CONNECT_REQ, quindi, la chiavetta è in grado di eseguire a sua volta il corretto algoritmo di frequency hopping e saltare da un canale all'altro contemporaneamente ai due dispositivi connessi, permettendo l'intercettazione di tutti i pacchetti trasmessi.

2.3.4 Scapy

Scapy è un software basato su Python utilizzabile sia mediante shell, sia come libreria esterna da importare in uno script [10]. Si tratta di uno strumento essenziale per la riuscita degli attacchi proposti nel progetto in quanto offre la possibilità di costruire pacchetti di rete manualmente (forging). Scapy permette anche di inviare i pacchetti costruiti e ricevere (e decodificare) i pacchetti provenienti da dispositivi esterni. Può essere visto, quindi, come un software analogo a Wireshark o tcpdump, ma con alcune funzionalità aggiuntive.

Capitolo 3

Implementazione degli attacchi

Il progetto di tesi mira alla replicazione di alcuni degli attacchi proposti alla sezione 2.2, in modo tale da dimostrarne la fattibilità implementando concretamente i vari passi necessari al completamento degli attacchi stessi. Si noti che, al fine di svolgere tutti gli esperimenti in maniera controllata, si rende necessaria la creazione di un ambiente isolato in modo da non sottoporre a nessun rischio altri dispositivi presenti nella zona: in ognuna delle simulazioni proposte, gli host della rete BLE sono costituiti da appositi dispositivi con sistema operativo basato su Linux o da microcontrollori ESP32. D'ora in poi, quando si farà riferimento al termine "dispositivo host", verrà considerato un dispositivo generico basato su sistema operativo Linux (computer, Raspberry PI, smartphone Android) oppure un microcontrollore ESP32, se non diversamente specificato. Si introducono, di seguito, gli attacchi che verranno approfonditi in questo capitolo:

- sviluppo di un beacon caratterizzato dall'invio di dati in formato non standard e tentativo di reverse-enegineering tramite Wireshark utilizzato in combinazione con la chiavetta Nordic;
- simulazione di un attacco MITM e successiva replicazione dei pacchetti in un attacco replay;

3.1 Nozioni base per l'utilizzo di Scapy

Si illustrano, in questa sezione, alcune delle più importanti funzionalità offerte da Scapy in modo da facilitare la comprensione dell'implementazione proposta per le varie parti del progetto.

3.1.1 Manipolazione pacchetti

Scapy offre un modo per la costruzione manuale di pacchetti piuttosto comodo. Occorre istanziare un oggetto per ogni layer del pacchetto in costruzione, con la possibilità di impostare i relativi attributi inserendo i corrispondenti parametri al momento della chiamata al costruttore dell'oggetto in questione. Scapy introduce un operatore dedicato alla costruzione del pacchetto finale a partire dalle istanze delle varie classi corrispondenti ai diversi layer: lo slash "/". In particolare, occorre concatenare, mediante questo operatore, tutti i livelli a partire da quello più basso dello stack: ogni oggetto incapsula il successivo. Nel caso specifico del Bluetooth, non è possibile costruire pacchetti manualmente a partire dai livelli dello stack facenti parte del controller, bensì occorre costruire comandi da inviare all'HCI: sarà poi il controller a completare le operazioni richieste ed inviare, successivamente, un pacchetto di risposta all'host.

```
1 HCI_Hdr()/HCI_Command_Hdr()/
2     HCI_Cmd_LE_Set_Scan_Enable(enable=True,filter_dups=False))
```

Algoritmo 3.1: Costruzione di un pacchetto di esempio in Scapy

Nell'esempio riportato sopra si costruisce un pacchetto con header standard per HCI definito come HCI_Hdr(); si incapsula, poi, l'header che distingue i pacchetti relativi a comandi per l'HCI; infine si specifica il vero e proprio comando da eseguire: in questo caso viene avviato il processo di scansione attiva e si segnala di non scartare i dispositivi di cui si ricevono i pacchetti di advertising, anche in presenza di duplicati.

3.1.2 Invio e ricezione dei pacchetti

Come anticipato precedentemente, Scapy ha la possibilità di comunicare direttamente con l'HCI corrispondente ad un determinato controller BLE. La comunicazione avviene tramite l'apertura di una socket tra lo script che utilizza le API di Scapy e l'HCI. Esistono vari tipi di socket offerti da Scapy [9] ma, ai soli fini della comprensione del progetto svolto, se ne illustrano in breve solamente due:

- *BluetoothHCISocket*: Scapy non ha un controllo esclusivo sull'HCI. Questo significa che Scapy può normalmente inviare comandi all'HCI ma è possibile che, simultaneamente, altri processi attivi sull'host comunichino a loro volta con l'HCI.
- *BluetoothUserSocket*: in questo caso Scapy ha controllo esclusivo sull'HCI.

Questa distinzione si rende necessaria dato che l'implementazione dello stack Bluetooth in ambiente Linux, denominata BlueZ, gestendo tutti gli aspetti e le interazioni tra sistema operativo e controller, potrebbe anche interferire con Scapy e produrre risultati

sperimentali inaspettati. Una volta istanziato l'oggetto relativo alla socket, dapprima occorre costruire i pacchetti come riportato nel paragrafo precedente e, successivamente, è possibile utilizzare diversi metodi sull'oggetto socket per inviare pacchetti all'HCI ed, eventualmente, ricevere gli eventi sollecitati dal controller.

3.2 Beacon personalizzato con Manufacturer Specific Data

Questa parte del progetto dimostra l'impossibilità di trasmettere i dati in maniera sicura in caso di metodo di comunicazione basato su advertising. In particolare, si vuole dimostrare che, pur utilizzando il tipo di dato "Manufacturer Specific Data" (sez. 2.2.1) e codificando i dati trasmessi in un formato non standard, tramite tecniche di reverse engineering è possibile leggere in chiaro il contenuto del pacchetto. La simulazione di questa situazione è così costituita:

- Si impiega un dispositivo host per la creazione di un beacon. L'obiettivo del beacon è di costruire (ed inviare) pacchetti di advertising che facciano uso del tipo "Manufacturer Specific Data" utilizzando le funzioni offerte da Scapy. Non è importante la natura dei dati inviati: in questo contesto si vuole dimostrare che, tramite tecniche di reverse engineering, la struttura interna dei dati potrebbe essere decodificata da un eventuale observer malevolo.
- Un altro dispositivo host viene utilizzato per la simulazione dell'utilizzatore legittimo del segnale prodotto dal beacon. Questo dispositivo, tramite un software sviluppato appositamente, ha la capacità di interpretare il formato non standard dei pacchetti di advertising ricevuti.
- Un terzo dispositivo, mediante utilizzo della chiavetta Nordic in accoppiata con Wireshark, può filtrare ed analizzare i pacchetti inviati dal beacon. Il formato del beacon è, in realtà, molto semplice da interpretare e, per questo motivo, non si rendono necessari ulteriori strumenti software.

Viene illustrata, di seguito, l'implementazione del beacon, oltre ad una breve descrizione del funzionamento del dispositivo designato come utilizzatore legittimo del segnale prodotto dal beacon. Infine, viene dimostrato come, tramite un semplice processo di reverse engineering, sia possibile derivare la struttura interna dei pacchetti di advertising trasmessi dal beacon con l'aiuto di Wireshark.

3.2.1 Implementazione del beacon

L'implementazione del beacon prevede una fase di costruzione del pacchetto di advertising e una fase di invio dello stesso. In particolare, il beacon sviluppato simula un

cardiofrequenzimetro: a partire da un valore iniziale del battito, si calcola il successivo sottraendo o aggiungendo una quantità predefinita, per ogni secondo. Lo scopo del beacon è, quindi, quello di costruire un nuovo pacchetto di advertising ogni secondo ed informare il controller tramite un apposito comando per l'HCI.

Analizzando la struttura del payload di advertising (figura 2.12) e la struttura del tipo Manufacturer Specific Data (figura 2.18), è possibile definire, grazie a Scapy, il formato del pacchetto di advertising.

```
LENGTH = b'\x05'
MANUFACTURER_SPECIFIC_DATA_AD_TYPE = b'\xff'
COMPANY_ID = b'\xfc\x91'
...
AD_data = LENGTH + MANUFACTURER_SPECIFIC_DATA_AD_TYPE + COMPANY_ID +
    bytes.fromhex(hearthbeat)
```

Algoritmo 3.2: Composizione del pacchetto di advertising

Il codice riportato definisce, byte per byte, i vari campi di cui è composto il pacchetto di advertising, in particolare:

- *Length* corrisponde alla dimensione del dato che, si ricordi, comprende sia il dato stesso che il suo tipo.
- *Manufacturer Specific Data* definisce il tipo del dato, specificato all'interno della documentazione relativa agli *Assigned Numbers*[4].
- *Company ID* corrisponde al campo, definito da standard, riservato al codice univoco assegnato ad ogni produttore registrato al Bluetooth SIG. Nel caso del beacon implementato, il valore di questo campo riveste, ovviamente, un ruolo fittizio. Il codice utilizzato in questo caso identifica la Samsung Electronics Co., Ltd.

Occorre notare che i campi elencati di sopra non variano durante l'attività del beacon e vengono, per questo motivo, definiti come costanti. I dati rilevanti da trasmettere, in questo caso costituiti esclusivamente dal battito cardiaco, sono composti da due byte e possono essere inseriti in coda ai precedenti campi, esattamente come definito da standard. La differenza è data dal fatto che il valore del battito cardiaco, come anticipato, varia una volta al secondo e, di conseguenza, il valore di "AD_data" varia di soli due byte per ciclo.

3.2.2 Implementazione dell'utilizzatore legittimo del beacon

Lo script di simulazione dell'utente legittimo del beacon utilizza una libreria Python denominata "Bleak" [6], dedicata proprio all'integrazione della tecnologia BLE all'inter-

no di programmi Python. In breve, lo script sviluppato entra in modalità di scansione analizzando periodicamente i pacchetti di advertising ricevuti, soffermandosi in caso uno di questi abbia il campo "Company ID" corrispondente a quello ricercato. Ai fini del funzionamento della comunicazione tra beacon e user è, infatti, necessario che il codice univoco della compagnia risulti uguale: lo script sviluppato tiene in considerazione solo i pacchetti di advertising classificati come appartenenti a Samsung Electronics Co., Ltd.

```
if company_id == int.from_bytes(COMPANY_ID, byteorder='little'):
    hearthbeat = data.hex()[0:4]
    hearthbeat_value = int(hearthbeat, 16)
```

Algoritmo 3.3: Decodifica dei dati ricevuti dal beacon

Come riportato nel codice presentato, quindi, una volta aver analizzato il produttore assegnato al pacchetto, si procede a quella che si può definire una vera e propria decodifica, seppur molto semplice. Lo user legittimo è a conoscenza, infatti, del formato utilizzato dal beacon per l'invio dei dati e preleva i primi due byte corrispondenti al valore del battito cardiaco.

3.2.3 Sniffing della rete tramite chiavetta Nordic

Durante la fase di advertising descritta, nessun altro dispositivo dovrebbe essere in grado di interpretare i dati contenuti nei pacchetti di advertising provenienti dal beacon. Si potrebbe, quindi, pensare che la protezione dei dati sia garantita in questo modo ma, come ampiamente anticipato, la natura broadcast dei pacchetti di advertising non impedisce a dispositivi terzi di leggere i dati trasmessi. Si precisa che, per lo svolgimento di questa simulazione, viene utilizzata la chiavetta Nordic in combinazione con Wireshark per svolgere un'analisi dei pacchetti ma, proprio a causa della natura broadcast dell'advertising, si potrebbe utilizzare come interfaccia di cattura un generico controller BLE. Avendo a disposizione i dati in chiaro, quindi, l'unico problema rimane

No.	Time	Source	Data
1	0.000000	f0:03:8c:56:12:42	05 ff 91 fc 00 59
2	3.865665	f0:03:8c:56:12:42	05 ff 91 fc 00 52
3	5.137545	f0:03:8c:56:12:42	05 ff 91 fc 00 54
4	8.983816	f0:03:8c:56:12:42	05 ff 91 fc 00 52
5	11.551984	f0:03:8c:56:12:42	05 ff 91 fc 00 52
6	14.124465	f0:03:8c:56:12:42	05 ff 91 fc 00 51

Tabella 3.1: Pacchetti di advertising visualizzati da Wireshark

quello dell'interpretazione degli stessi: proprio a questo scopo si tenta di decodificare la struttura dei dati analizzando i pacchetti grazie a Wireshark. La tabella 3.1 mostra una serie di pacchetti di advertising filtrati per "Advertising Address", corrispondente a quello pubblicato dal beacon stesso. In questo caso d'esempio, molto semplice, è intuitivo notare come tutti i dati di advertising contengano una stringa di N byte fissa. Gli ultimi due byte, invece, variano tra un pacchetto e l'altro: si può affermare, con una discreta certezza, che il cardiofrequenzimetro riservi gli ultimi due byte proprio alla trasmissione del battito cardiaco.

3.3 Attacchi MITM e Replay

L'obiettivo di questa parte del progetto è quello di implementare un vero e proprio attacco di tipo MITM, con possibilità di effettuare successivi attacchi di tipo replay. Come spiegato nella sezione 2.2.4, la forma di comunicazione più vulnerabile ai suddetti tipi di attacco è rappresentata dalla connessione non sicura (priva di pairing): non effettuando alcuna forma di crittografia, l'unica misura di (pseudo) sicurezza è fornita dal frequency hopping. La simulazione descritta di seguito vuole dimostrare, quindi, in che modo superare la barriera del frequency hopping per leggere in chiaro l'intera comunicazione. Anche in questo caso vengono impiegati tre diversi dispositivi host, in particolare:

- Il primo dispositivo host riveste il ruolo del server legittimo e si comporta come tale offrendo alcuni servizi GATT ed eseguendo un processo di advertising iniziale per permettere ad altri dispositivi di poter richiedere la connessione.
- Un altro dispositivo host riveste, invece, il ruolo del client legittimo con intenzione di avviare una connessione con il server legittimo.
- Un ulteriore dispositivo riveste, chiaramente, il ruolo dell'attaccante. La peculiarità di questo dispositivo è data dal fatto che, pur possedendo due distinti controller BLE, è costituito comunque da un unico host che, tramite le corrispondenti HCI può comunicare con entrambi i controller. L'obiettivo di questo dispositivo è quello di ricevere i pacchetti provenienti dal client, analizzarli tramite Scapy ed eventualmente memorizzarli per l'esecuzione di un futuro attacco replay. Utilizzando le funzioni offerte da Scapy, occorre anche "re-impacchettare" i dati ricevuti da un lato per inoltrarli all'altro, eventualmente previa modifica con conseguente ricalcolo del CRC.

Di seguito, si introducono dapprima i dispositivi utilizzati per ricoprire i diversi ruoli e, successivamente, si mostra l'implementazione dell'attacco MITM suddividendo il processo nelle varie fasi descritte a sezione 2.2.4. Infine, viene dettagliata l'implementazione dell'attacco replay, evidenziandone le differenze rispetto al MITM.

3.3.1 Client e server legittimi

Per la replicazione di un attacco MITM è, ovviamente, necessaria la presenza di un client e di un server legittimi che abbiano intenzione di instaurare una connessione. Nell'ambito della simulazione proposta in questo progetto di tesi il ruolo del server viene svolto da un microcontrollore ESP32-C6 appositamente programmato per fornire un servizio di misurazione del battito cardiaco (simulato). Il software caricato sul microcontrollore è stato sviluppato da Espressif ed è disponibile pubblicamente nella sezione "esempi" della repository GitHub proprietaria [5]. Si sottolinea che, ai fini sperimentali, non è importante la natura dell'applicativo eseguito dall'ESP32: è sufficiente che il server offra un servizio con una o più caratteristiche che, a loro volta, permettano la scrittura e/o la lettura da parte di un eventuale client.

```
1 static const uint16_t GATTS_SERVICE_UUID_TEST      = 0x00FF;
2 static const uint16_t GATTS_CHAR_UUID_TEST_A       = 0xFF01;
3 static const uint16_t GATTS_CHAR_UUID_TEST_B       = 0xFF02;
4 static const uint16_t GATTS_CHAR_UUID_TEST_C       = 0xFF03;
```

Algoritmo 3.4: Definizione del servizio e relative caratteristiche

Per quello che riguarda il client, invece, si utilizza l'applicazione per smartphone "nRF Connect" sviluppata da Nordic. nRF Connect si rivela estremamente comoda per questo caso d'uso in quanto permette di scansionare la rete BLE e, una volta trovato il server GATT di interesse, avviare una connessione con esso. Offre, inoltre, la possibilità di visualizzare i servizi gestiti dal server e interagire con essi a basso livello tramite pacchetti ATT.

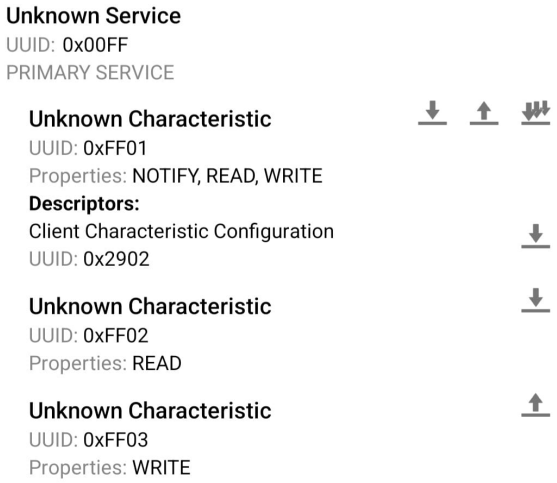

Figura 3.1: Schermata presentata da nRF Connect una volta avvenuta la connessione con il server implementato sull'ESP32

Nella figura 3.1 è riportata una schermata dell'applicazione nRF Connect che, una volta connessa al server implementato su ESP32, permette di visualizzare ed interagire con le varie caratteristiche disponibili. Di controparte, il codice riportato nell'algoritmo 3.4, rappresenta la vera e propria definizione del servizio offerto dal server: in questo caso il servizio viene identificato da UUID pari a 0x00FF ed è costituito da tre diverse caratteristiche, ognuna delle quali identificata da un UUID univoco.

3.3.2 Architettura del software sviluppato

Ai fini di facilitare la comprensione della fase implementativa degli attacchi proposti, occorre dapprima specificare alcuni dettagli sull'architettura del software sviluppato per l'esecuzione effettiva degli attacchi. Così come anticipato nella sezione 2.2.4, un attacco Man In The Middle è caratterizzato dall'apertura di due connessioni distinte e simultanee: una tra client legittimo e server malevolo e l'altra tra client malevolo e server legittimo. Si noti che entrambi i ruoli "malevoli" vengono, in realtà, svolti dallo stesso dispositivo intermediario mediante l'utilizzo di due diversi controller Bluetooth. Per implementare questo meccanismo tramite Scapy è sufficiente utilizzare due BluetoothUserSocket. In particolare, ci si riferirà a "Client socket" e "Server socket" per indicare il canale dedicato alla comunicazione tra l'host e il controller utilizzato per comunicare, rispettivamente, con il client legittimo e il server legittimo.

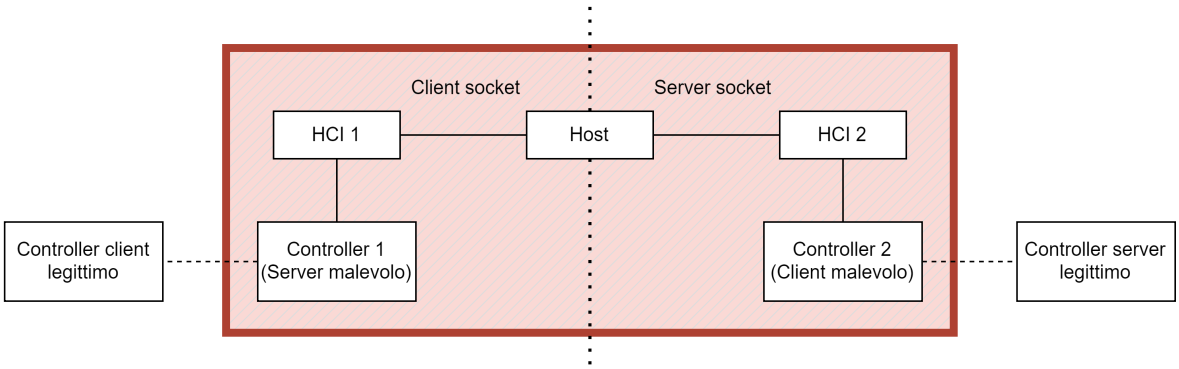

Figura 3.2: Gestione delle connessioni tramite due diverse socket

Il software si compone inoltre di diversi moduli:

- Un modulo dedicato alla fase di scansione della rete, utilizzato da entrambi gli attacchi in modo da individuare il server vittima.
- Un modulo costituito da alcune funzioni utili a semplificare la comunicazione tramite socket e a decodificare i vari pacchetti BLE ricevuti dai controller.

- Il cuore pulsante degli attacchi, modulo costituito da tutte le funzioni che implementano concretamente le diverse fasi degli attacchi. Occorre sottolineare che l'attacco replay include, in realtà, alcune delle funzioni sviluppate per l'attacco MITM: la principale differenza è data dalla possibilità di importare pacchetti ATT precedentemente memorizzati e inviarli al server vittima dell'attacco.

Per l'esecuzione effettiva degli attacchi si utilizzano due diversi script, uno per ogni tipo di attacco. Questi script possono essere visti come moduli "driver": importano i moduli descritti precedentemente, predispongono le socket necessarie, eseguono la fase di scansione e, anche mediante alcuni parametri impostati dall'utente, avviano l'attacco.

3.3.3 Prima fase: individuazione del server da attaccare

In questa fase, sia che si tratti di un attacco replay che di un MITM, l'obiettivo è quello di entrare in modalità di scansione in modo da poter ricevere i pacchetti di advertising dei diversi dispositivi presenti in rete e, di conseguenza, selezionare un server GATT da attaccare. Occorre, quindi, inviare alcuni comandi all'interfaccia del controller dedicato alla gestione della comunicazione con il server legittimo: si impiega la server socket.

```
HCI_Hdr()/HCI_Command_Hdr()/HCI_Cmd_LE_Set_Scan_Parameters(type=0),
HCI_Hdr()/HCI_Command_Hdr()/HCI_Cmd_LE_Set_Scan_Enable(enable=1,
    filter_dups=1)
```

Algoritmo 3.5: Attivazione modalità di scansione passiva

Il codice riportato sopra, mostra la costruzione di due diversi comandi destinati all'HCI: il primo utile all'impostazione del tipo di scansione da eseguire (in questo caso si tratta di una scansione passiva); il secondo, invece, responsabile dell'attivazione effettiva della modalità di scansione.

```
while True:
    incoming_pkt = recv_packet(socket,0)
    if not incoming_pkt:
        break
    if HCI_LE_Meta_Advertising_Reports in incoming_pkt:
        adv_report = incoming_pkt[HCI_LE_Meta_Advertising_Reports]
        for report in adv_report.reports:
            devices[report.addr] = report.data
```

Algoritmo 3.6: Analisi report di advertising

Una volta disattivata la modalità di scansione, si analizza la coda di eventi generati dal controller: durante la scansione eseguita, per ogni intervallo di tempo predefinito, il controller genera un evento denominato "Advertising_Reports" che fornisce maggiori informazioni sugli eventuali pacchetti di advertising ricevuti. Nel codice riportato sopra vengono prelevati, uno ad uno, tutti gli eventi di questo tipo in modo da costruire un dizionario che abbina ad ogni indirizzo i dati contenuti nel pacchetto di advertising corrispondente. A partire dal dizionario costruito mediante questa funzione, l'utente ha la possibilità di scegliere il server vittima dell'attacco.

3.3.4 Seconda fase: connessione al server e riproduzione dell'advertising

Una volta selezionato il server vittima, si procede ad inviare una richiesta di connessione (CONNECT_REQ): questa operazione provoca la terminazione della fase di advertising del server legittimo e impedisce che quest'ultimo possa essere individuato da altri client. L'esecuzione di questo passo tramite Scapy è di estrema facilità e prevede l'invio di un apposito comando all'HCI utilizzando la server socket, specificando come parametro l'indirizzo del server vittima.

In questo contesto, si rende utile introdurre brevemente il concetto di "handle" della connessione. Si tratta di un numero intero utilizzato dai controller per identificare univocamente una connessione stabilita. Si noti che la "handle della connessione" rappresenta un concetto del tutto indipendente rispetto alla "handle dell'attributo" descritta a sezione 2.1.2 relativamente al protocollo ATT. La handle della connessione si rivela fondamentale soprattutto nel caso dell'attacco MITM: per inoltrare un pacchetto da un capo all'altro della connessione è necessario, ogni volta, prelevare i dati contenuti nel pacchetto ricevuto ed inviare un apposito comando all'HCI contenente sia i dati appena prelevati, sia la handle della connessione a cui partecipa il dispositivo destinatario del pacchetto. Risulta quindi necessario riuscire a prelevare e successivamente memorizzare le handle delle connessioni instaurate.

```
while True:
    incoming_pkt = recv_packet(self.server_sock,3)
    if not incoming_pkt:
        break
    if hasattr(incoming_pkt, 'handle'):
        server_handle = incoming_pkt.handle
```

Algoritmo 3.7: Individuazione della handle della connessione

Il codice riportato sopra viene eseguito successivamente alla richiesta di connessione indirizzata al server legittimo. Il suo scopo è quello di analizzare i pacchetti in entrata,

corrispondenti agli eventi generati dal controller, in cerca di eventuali pacchetti contenenti una handle. In caso di connessione avvenuta con successo, infatti, il controller assegna una handle in automatico: ogni successivo pacchetto proveniente dal server provoca la sollecitazione di un evento da parte del controller in cui, oltre ai dati ricevuti, si comunica anche la handle corrispondente. Proprio grazie a questo meccanismo si riesce a prelevare la handle della connessione: è sufficiente che il server invii un pacchetto di conferma di connessione affinché la funzione riportata in figura individui e memorizzi la handle corrispondente.

```
HCI_Hdr()/HCI_Command_Hdr()/HCI_Cmd_Reset(),
HCI_Hdr()/HCI_Command_Hdr()/
    HCI_Cmd_LE_Set_Random_Address(address=self.server_paddr),
HCI_Hdr()/HCI_Command_Hdr()/
    HCI_Cmd_LE_Set_Advertising_Parameters(oatype=1),
HCI_Hdr()/HCI_Command_Hdr()/
    HCI_Cmd_LE_Set_Advertising_Data(data=adv_data),
HCI_Hdr()/HCI_Command_Hdr()/HCI_Cmd_LE_Set_Advertise_Enable(enable=1)
```

Algoritmo 3.8: Comandi necessari all'emulazione dell'advertising del server

A questo punto, si ha la prima delle due connessioni richieste per l'esecuzione dell'attacco, in particolare si può affermare che il server vittima sia stato "catturato". Manca, ovviamente, la cattura del client vittima e, proprio per questo motivo, è necessario replicare esattamente i pacchetti di advertising precedentemente ricevuti dal server legittimo. Questa operazione, descritta con il nome di "device spoofing" e precedentemente introdotta nella sezione 2.2.3, si rivela di fondamentale importanza per la corretta riuscita dell'attacco e viene eseguita previo invio di alcuni comandi in maniera sequenziale:

1. *Reset*, utile ad eliminare eventuali impostazioni precedenti riguardanti l'HCI.

2. *Set Random Address*, comando utile ad impostare un indirizzo di tipo "random" (sezione 2.1.3). In questo caso occorre impostare il valore dell'indirizzo in modo che sia uguale all'indirizzo pubblicato dal server legittimo.

3. *Set Advertising Parameters*, comando volto all'impostazione di alcuni parametri relativi all'advertising. Nel caso di un attacco MITM, questo comando si rende necessario per specificare la natura dell'indirizzo pubblicato: come detto prima, si tratta di un indirizzo random.

4. *Set Advertising Data*, fondamentale per l'emulazione vera e propria del pacchetto di advertising proveniente dal server legittimo. Tramite questo comando è possibile definire i dati da pubblicare che chiaramente, nel caso di un attacco MITM, corrispondono ai dati precedentemente ricevuti dal server legittimo.

5. *Set Advertising Enable*, comando responsabile dell'attivazione effettiva della fase di advertising.

Si noti che tutti i pacchetti definiti devono essere inviati tramite client socket, in quanto destinati al controller dedicato alla gestione della connessione con il client vittima.

3.3.5 Terza fase: attesa del client e realizzazione dell'attacco

Una volta conclusa anche la seconda fase, si ha un contesto in cui è presente una connessione tra l'attaccante e il server legittimo e, simultaneamente, il controller dell'attaccante dedicato alla gestione della connessione col client legittimo è in fase di advertising. In questa situazione, quindi, gli eventuali client in fase di scansione sono impossibilitati ad individuare il server legittimo ma, a loro insaputa, individuano invece un server completamente identico, corrispondente al server malevolo. La fase di attesa e cattura del client viene eseguita in maniera simile a come riportato nell'algoritmo 3.7, con la differenza che, invece che cercare un qualsiasi pacchetto che contenga una handle, si analizza la coda di eventi generati dal controller per individuare un pacchetto di tipo "Connection Complete": a partire da questo pacchetto si ricavano tutti i dati necessari alla gestione della connessione (handle e indirizzo del client). Si noti che la coda di eventi analizzati è quella relativa al controller assegnato al server malevolo, si opera perciò sulla client socket.

Una volta stabilite entrambe le connessioni si può procedere alla fase di attacco vero e proprio. Il meccanismo responsabile dell'operazione di inoltro da un capo all'altro della connessione si basa interamente sulle categorie di PDU definite dal protocollo ATT e approfondite nella sezione 2.1.2. In particolare l'algoritmo proposto è composto dai seguenti passaggi:

1. Una volta stabilita la connessione col client legittimo, lo script eseguito dall'attaccante rimane in attesa di ricevere pacchetti ATT. Questa operazione viene implementata mettendosi in ascolto sulla client socket.

2. Una volta ricevuto un evento di ricezione di un pacchetto ATT, lo script decodifica il pacchetto e individua la categoria di appartenenza. Infine, inoltra il pacchetto al server legittimo usufruendo della server socket.

3. Se il pacchetto inoltrato è categorizzato come "ATT Command", la socket di ascolto rimane quella del client. Al contrario, in caso di "ATT Request", lo script rimane in ascolto sulla server socket. Questo meccanismo è, in sostanza, il cuore dell'intero attacco: in presenza di un Command infatti il client non ha bisogno di nessuna risposta da parte del server e, per questo motivo, lo script si limita ad inoltrare il comando stesso al server legittimo; in caso si tratti di una Request, invece, il client legittimo si aspetta di ricevere un "ATT Response" dal server e,

proprio per questo motivo, occorre bloccare lo script e rimanere in ascolto sulla server socket, in modo tale da inoltrare al client legittimo l'eventuale pacchetto di risposta proveniente dal server legittimo. Una volta ricevuto (e inoltrato al client legittimo) il pacchetto di Response, si ritorna al punto 1.

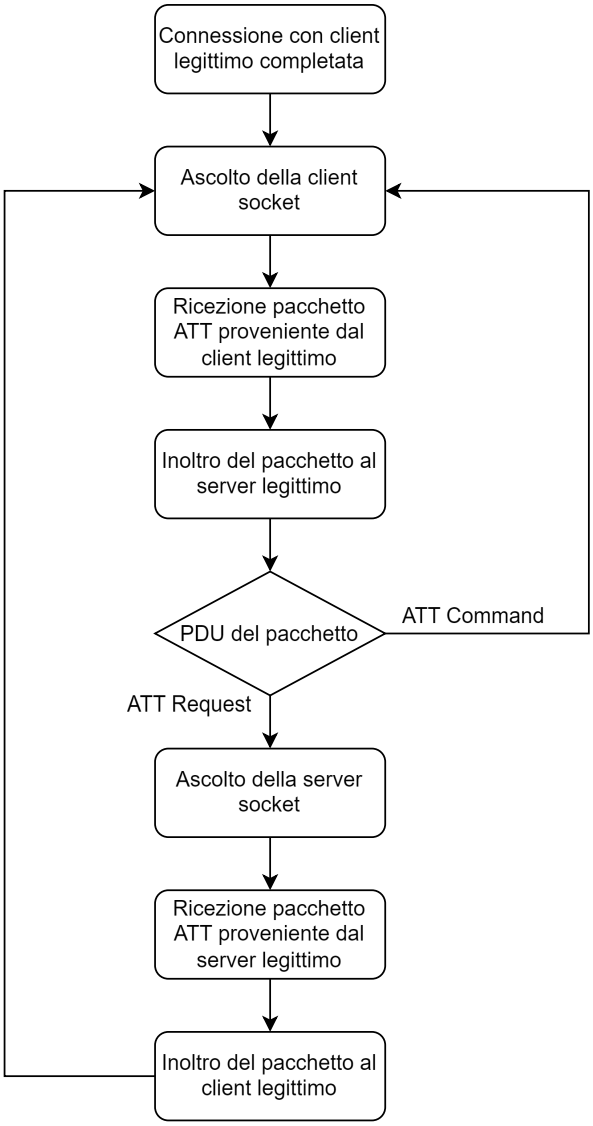

Figura 3.3: Diagramma di flusso dell'algoritmo proposto per l'attacco MITM

Occorre sottolineare che, ai fini di semplificare l'implementazione, il software proposto non gestisce le altre due categorie di messaggi ATT: Notification e Indication. Queste due categorie di pacchetti ATT, comunque, possono essere considerate analoghe a Command e Request, con l'unica differenza data dal fatto che il primo messaggio, in entrambi i casi, viene inviato dal server invece che dal client. L'algoritmo implementato non sarebbe quindi capace di gestire queste situazioni in quanto, impostando come socket di ascolto iniziale quella corrispondente al client, verrebbero scartati eventuali messaggi provenienti dal server. Una possibile soluzione al problema potrebbe essere

data dall'implementazione dello stesso algoritmo in maniera "specchiata". In particolare, la fase di ascolto iniziale potrebbe essere eseguita sulla server socket invece che sulla client socket. Abbinando l'algoritmo originale a quello in versione "specchiata", e assegnando ognuno dei due ad un diverso thread, sarebbe possibile gestire ogni categoria di pacchetto ATT.

```
if self.currently_waiting == self.client_sock:
    if is_ATT_Request(incoming_pkt):
        self.ATT_forward(self.server_sock, incoming_pkt)
    elif ATT_Write_Command in incoming_pkt:
        self.ATT_forward(self.server_sock, incoming_pkt)
else:
    if is_ATT_Response(incoming_pkt) and self.currently_waiting ==
       self.server_sock:
        self.ATT_forward(self.client_sock, incoming_pkt)
```

Algoritmo 3.9: Categorizzazione dei pacchetti ATT ricevuti

3.3.6 Implementazione dell'attacco replay

Come specificato a sezione 2.2.4, per condurre un attacco di tipo replay è necessaria, in primis, l'esecuzione di un attacco MITM. A partire da un attacco Man In The Middle è possibile eseguire un replay attack in diversi modi. In particolare, è possibile inviare messaggi duplicati durante l'esecuzione dell'attacco MITM stesso, oppure potrebbe essere utile memorizzare i pacchetti inviati dal client legittimo al server legittimo in modo da poterli riutilizzare in una connessione successiva. L'implementazione proposta in questo progetto di tesi si basa sul secondo metodo. Parte del processo per l'esecuzione dell'attacco replay è condivisa con l'attacco MITM, in particolare la prima fase (sezione 3.3.3) e la fase di connessione al server legittimo, descritta nella sezione 3.3.4. Si noti che la tipologia di attacco replay proposto non richiede l'utilizzo di due controller: è sufficiente che il client malevolo disponga di un singolo controller per instaurare la connessione con il server legittimo. Inoltre, data la necessità di una singola connessione, è sufficiente utilizzare una sola socket per interfacciarsi con il controller tramite Scapy.

Una volta stabilita la connessione col server, lo script importa una serie di pacchetti precedentemente memorizzati durante un attacco MITM e permette all'utente di selezionare il messaggio da inviare al server. In questo attacco, il meccanismo di ascolto della socket proposto per l'attacco MITM risulta assai semplificato: una volta selezionato un pacchetto da inviare, lo script individua la categoria di appartenenza e, in caso di Command, una volta inviato il pacchetto ripropone immediatamente il menu di scelta all'utente mentre, in caso di Request, è necessario che lo script rimanga in ascolto sulla socket aperta precedentemente. Quest'ultimo passaggio risulta fondamentale in quanto permette di individuare eventuali scenari di errore in cui, per qualche

motivo, il server non invia una Response. Una volta ricevuto il pacchetto di Response, il programma ripropone la scelta all'utente.

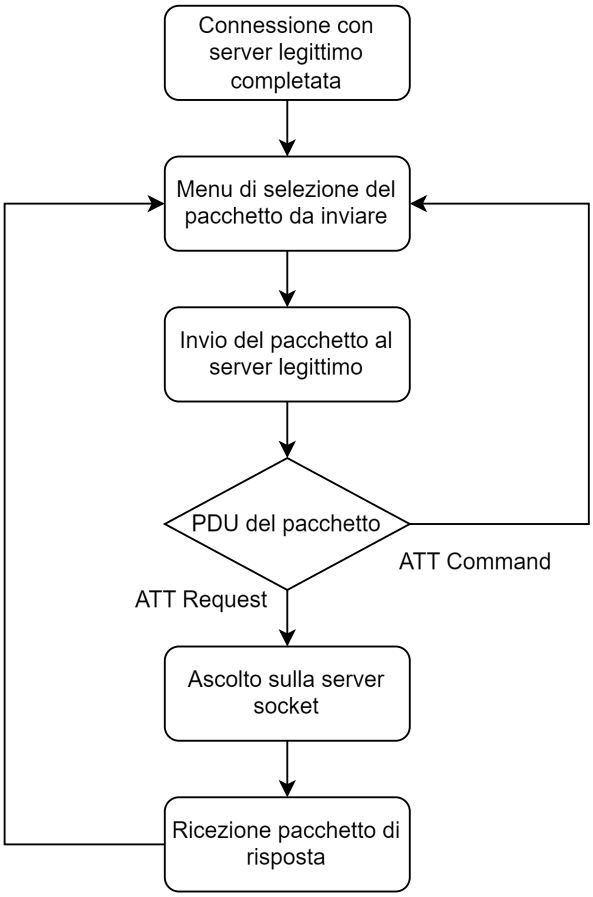

Figura 3.4: Diagramma di flusso dell'algoritmo proposto per l'attacco replay

Per quello che riguarda il caso specifico dell'implementazione proposta, i pacchetti inviati dal client vengono memorizzati dal metodo "ATT_Forward", dedicato all'inoltro dei messaggi tra un capo e l'altro della connessione durante l'attacco MITM (algoritmo 3.9). In particolare, il codice riportato nell'algoritmo 3.10, rappresenta la sezione del metodo di inoltro dedicata all'invio dei messaggi tramite server socket. Questa sezione gestisce, quindi, i messaggi che vengono inviati dal client legittimo. Si osservi che il pacchetto inviato viene memorizzato in una lista, la quale viene salvata su un file ".pcap" una volta terminato l'attacco in modo tale che lo script utilizzato per l'esecuzione dell'attacco replay possa importare tale file e ricostruire la lista di pacchetti. Inoltre, il passo di codice riportato è utile anche per comprendere appieno il funzionamento di decomposizione e ricomposizione dei pacchetti da inoltrare nel corso dell'attacco MITM. Tale meccanismo è costituito dai seguenti passi:

1. Ricezione del pacchetto ATT da parte del controller A e generazione evento apposito per notificare l'host intermediario. L'evento generato contiene i dati trasportati dal pacchetto ATT ricevuto.

2. I dati del pacchetto ATT vengono prelevati dall'evento. Si noti la scrittura "pkt[L2CAP_Hdr]": in questo modo vengono prelevati dall'evento solo i dati relativi ai layer dello stack appartenenti all'host. Osservando la figura 2.1, è possibile notare che occorre selezionare il livello L2CAP che, a sua volta, incapsula tutti gli altri dati dei layer superiori.

3. In ultimo, per inviare i dati appena prelevati all'altro capo della connessione, occorre costruire un comando destinato all'HCI del controller B e contenente i dati stessi. Si noti, in questo passaggio, anche la definizione della handle.

```
1 if recipient == self.server_sock:
2     to_send =
          HCI_Hdr()/HCI_ACL_Hdr(handle=self.server_handle)/pkt[L2CAP_Hdr]
3     self.server_sock.send(to_send)
4     self.client_sent_pkts.append(to_send)
```

Algoritmo 3.10: Sezione della funzione ATT_Forward

Capitolo 4

Analisi della perdita di pacchetti in fase di advertising e connessione

L'ultima parte del progetto di tesi consiste nell'analisi del processo di advertising, e successiva richiesta di connessione, dal punto di vista della selezione del canale. In particolare, in questa sezione del progetto si vogliono approfondire le ripercussioni del meccanismo di salto tra i tre canali (37,38,39) dal punto di vista della perdita di pacchetti. Come descritto nella sezione 2.1.7, per avviare una connessione occorre che il client invii un pacchetto di CONNECT_REQ sullo stesso canale su cui è stato ricevuto il pacchetto di advertising da parte del server. Dato che i canali sono tre e la chiavetta Nordic salta continuamente tra i tre canali è possibile, però, che si verifichino situazioni in cui il pacchetto CONNECT_REQ non venga correttamente ricevuto dalla chiavetta, con conseguente impossibilità di seguire la connessione corrispondente. L'obiettivo di questa fase del progetto è proprio quello di definire, empiricamente, la probabilità che un pacchetto di richiesta di connessione venga correttamente ricevuto dalla chiavetta Nordic.

4.1 Descrizione del problema e introduzione all'ambiente di test

Come descritto alla sezione 2.1.6 , durante la fase di advertising un dispositivo BLE (configurato in maniera standard) invia pacchetti pubblicitari in rete trasmettendoli sui canali 37,38 e 39, in maniera round-robin. L'operazione complementare viene compiuta, invece, da eventuali dispositivi in fase di scanning della rete. Difatti, un controller BLE (configurato in maniera standard) rimane in ascolto sui canali 37,38 e 39, alternando tra i tre sequenzialmente così come avviene nel processo di advertising. Il fatto che un controller non abbia la capacità di ascoltare i tre canali di advertising simultaneamente, però, introduce la possibilità che un pacchetto di advertising venga

perso. Allo stesso modo, in fase di richiesta di connessione, esiste l'eventualità che un dispositivo periferico non riceva correttamente il pacchetto di CONNECT_REQ, inviato anch'esso su uno dei tre canali riservati all'advertising come descritto alla sezione 2.1.7. Si tratta, ovviamente, di un aspetto minore relativo alla tecnologia BLE ma che risulta comunque utile tenere a mente per far fronte ad eventuali operazioni di troubleshooting della rete.

Anche la chiavetta Nordic, per eseguire le operazioni di sniffing della rete, possiede un controller BLE e, come tutti gli altri dispositivi, è passibile di perdita di pacchetti. In questa parte del progetto si vuole quindi condurre uno studio, dal punto di vista statistico, relativo alla percentuale di pacchetti persi in fase di richiesta della connessione. L'ambiente di riferimento per questo tipo di sperimentazione è molto semplice:

- Un dispositivo client, gestito da un apposito script Scapy.
- Un ESP32 con ruolo di server, la cui configurazione risulta del tutto analoga a quella utilizzata per gli attacchi MITM e replay visti a sezione 3.3. Non è importante la natura dei servizi offerti: è necessaria solo la possibilità di ricevere una richiesta di connessione.
- Un terzo dispositivo in ascolto sulla rete, con l'ausilio di Wireshark e della chiavetta Nordic, si occupa dell'analisi dei pacchetti CONNECT_REQ ricevuti. Sulla base del numero dei pacchetti visualizzati da Wireshark e di quelli inviati dal dispositivo client si definisce l'esito dell'esperimento.

Lo script Scapy implementato sul client effettua, sequenzialmente, un certo numero N di tentativi di connessione al server. Fissato N, si effettuano diversi test modificando appositamente il numero D di chiavette Nordic utilizzate e il rispettivo insieme dei canali di ascolto C. Infatti Wireshark, integrato con l'apposito software prodotto da Nordic, permette di osservare passivamente la rete utilizzando più interfacce simultaneamente. Inoltre, sulla stessa interfaccia, è possibile selezionare l'insieme di canali da alternare ciclicamente durante la fase di scansione. Selezionando solo il canale 38, ad esempio, non viene ricevuto dalla chiavetta alcun advertising inviato sui canali 37 e 39 ma, al contempo, ci si può aspettare di ricevere la totalità degli advertising trasmessi sul canale 38: eliminando il fattore di salto tra un canale e l'altro, la chiavetta dovrebbe ricevere tutti i pacchetti di advertising trasmessi sull'unico canale ascoltato. Si sottolinea che, però, si tratta di una supposizione puramente teorica in quanto esistono altri fattori che provocano la perdita di pacchetti oltre al frequency hopping. Una possibile problematica in ambito Bluetooth può essere riscontrata, per esempio, nella distanza tra due dispositivi. Dal punto di vista puramente tecnico, lo script è implementato in maniera del tutto analoga al codice riportato nell'algoritmo 3.7 con l'unica differenza che lo stesso processo di connessione viene ripetuto, appunto, un numero N di volte. In particolare, il client tenta di connettersi al server predisposto e, una volta ricevuta

la handle, la connessione viene considerata attiva. In questo caso la handle risulta di fondamentale importanza in quando rappresenta l'unico parametro da abbinare all'apposito comando per l'HCI utile ad informare il server della disconnessione.

4.2 Risultati sperimentali

In questa sezione vengono illustrati i risultati dei test effettuati, fornendone sia una rappresentazione grafica sia una descrizione dei risultati ottenuti. Trattandosi di una tecnologia che utilizza la trasmissione via etere, il Bluetooth è caratterizzato da una maggiore perdita di pacchetti rispetto ad altre tecnologie operanti tramite cavo. Al fine di avere dei risultati il più veritieri possibili nei test effettuati, quindi, occorre eseguire un test preliminare per definire la reale distribuzione dei pacchetti nella fase di advertising e successiva connessione.

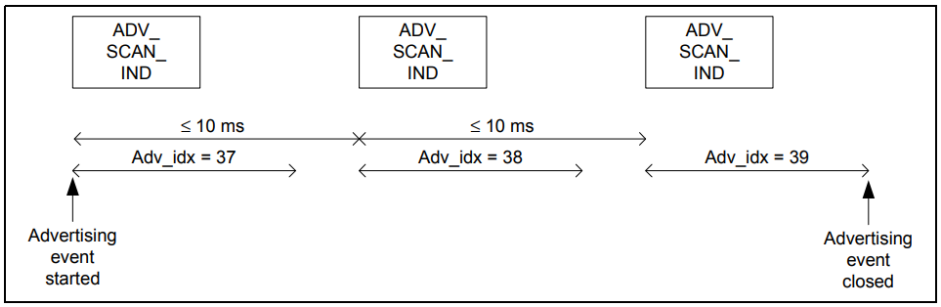

Figura 4.1: Advertising: tre canali alternati sequenzialmente

Test preliminare. Come illustrato nella figura 4.1, un dispositivo in fase di advertising con configurazione standard, invia un pacchetto di advertising per ogni intervallo di tempo predefinito, utilizzando tutti e tre i canali riservati all'advertising in maniera round-robin. Quindi, almeno teoricamente, effettuando il test implementato, ci si potrebbe aspettare che gli R pacchetti di CONNECT_REQ correttamente osservati dalla chiavetta siano equamente distribuiti sui tre canali. Il test preliminare proposto, in particolare, raggruppa un totale di tre test, effettuati affinché fosse possibile osservare la reale distribuzione dei pacchetti all'interno dell'ambiente utilizzato per lo svolgimento dei successivi test. Ogni test è caratterizzato dall'invio di 100 tentativi di connessione e l'osservazione viene condotta utilizzando una singola chiavetta Nordic. La differenza tra i tre test risiede nell'insieme dei canali C osservati: ogni test osserva un singolo canale (C_{test1}=37,C_{test2}=38, C_{test3}=39). Considerando che osservando un singolo canale si elimina il meccanismo di salto tra un canale e l'altro, si ritiene nulla la perdita di pacchetti relativi all'unico canale osservato. I risultati dei test, illustrati a figura 4.3, sono i seguenti:

- Test 1: N=100, D=1, $C = \{37\} \rightarrow R = 46$;

- Test 2: N=100, D=1, $C = \{38\} \rightarrow R = 25$;
- Test 3: N=100, D=1, $C = \{39\} \rightarrow R = 11$;

Chiaramente, come osservabile nei risultati, la distribuzione dei pacchetti di CONNECT_REQ ricevuti dalla chiavetta non è equivalente sui tre canali: in particolare, il 56% dei pacchetti è stato osservato sul canale 37, il 31% sul canale 38 e, il restante 13% sul canale 39. Questo risultato indica, per via sperimentale, che il client utilizzato per effettuare i test risponde al server prevalentemente sui canali 37 e 38. Si può concludere, quindi, che il canale 39 non viene utilizzato con la stessa frequenza per via di possibili interferenze presenti nell'ambiente di svolgimento dei test. I successivi test devono essere interpretati sulla base dei risultati del test preliminare.

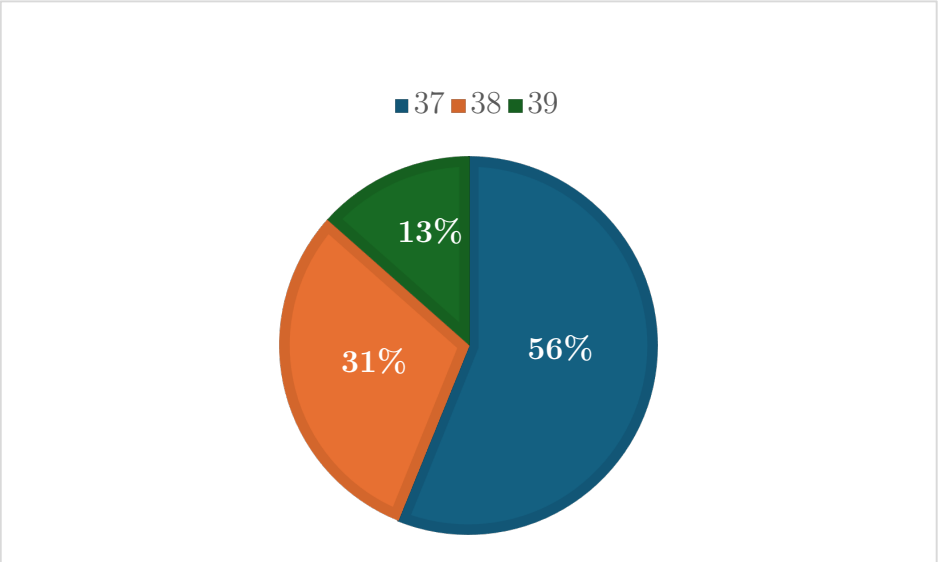

Figura 4.2: Distribuzione dei canali risultante dal test preliminare

Test A. Si illustra di seguito il primo test effettivo. Con questo test, si vuole dimostrare e quantificare sperimentalmente la perdita di pacchetti dovuta al frequency hopping. Data la distribuzione dei canali descritta sopra, questo test osserva esclusivamente i canali 37 e 38, tralasciando per il momento il canale 39. Il test proposto consiste nel confrontare il numero R di pacchetti correttamente ricevuti nel caso di utilizzo di una chiavetta singola con C={37,38} e nel caso di utilizzo di due chiavette distinte, una con C={37} e l'altra con C={38}. Quindi, i due sotto-test risultano così caratterizzati:

- Test A.1: N=100, D=1, $C = \{37, 38\} \rightarrow R = 60$;
- Test A.2: N=100, D=2
  - $C_{D1} = \{37\} \rightarrow R = 52$;

– $C_{D2} = \{38\} \rightarrow R = 24$;

I risultati di questo test confermano la perdita di una parte di pacchetti a causa del frequency hopping. Difatti, confrontando il test A.1 e il test A.2, la differenza tra il numero di pacchetti ricevuti risulta netta e ammonta a 16 pacchetti. Se, inoltre, si confrontano i dati di questo test con la distribuzione dei pacchetti caratteristica dell'ambiente utilizzato (figura 4.2) si può affermare che:

- Il test A.1 è riuscito ad osservare 60 pacchetti su 71, indicando una perdita di pacchetti del 16% causata dal frequency hopping.
- Il test A.2 è riuscito ad osservare 76 pacchetti su 71, confermando una perdita di pacchetti nulla dovuta all'assenza del frequency hopping.

Si precisa che nel caso del test A.2 il numero di pacchetti osservati è superiore al numero totale di pacchetti ricevuti nel test preliminare perchè, non potendo garantire un ambiente totalmente privo di interferenze, è possibile che la distribuzione dei pacchetti sui tre canali osservata nel test preliminare vari leggermente nel corso del tempo. Occorre quindi considerare un fattore di incertezza intrinseco in tutti i test eseguiti. Si può comunque affermare che il test A.2 ha ricevuto il 100% delle CONNECT_REQ inviate sui canali 37 e 38.

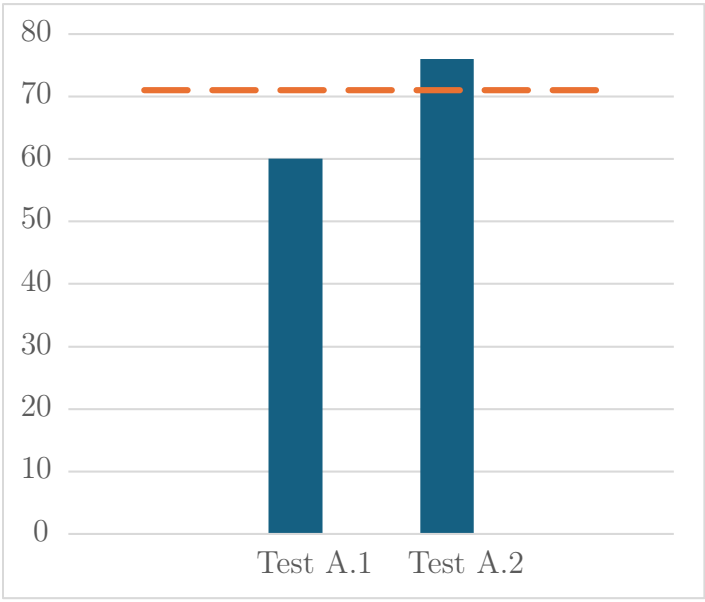

Figura 4.3: Il test A.2 raggiunge e supera il numero di pacchetti identificato dal test preliminare

Test B. In ultimo, viene proposto un test effettuato osservando tutti e tre i canali di advertising. Data la mancanza di una terza chiavetta, questo test estende il test riportato sopra: invece che osservare i tre diversi canali tramite tre chiavette si

utilizzano solo due chiavette, una con C=$\{37,38\}$ e l'altra con C=$\{39\}$. Per questo motivo, i dati ricavati da questo test non sono da intendersi come completamente privi di perdita dovuta al frequency hopping; dovrebbero comunque mostrare una perdita minore rispetto all'osservazione di tre canali con un'unica chiavetta. I due sotto-test risultano così caratterizzati:

- Test B.1: N=100, D=1, $C = \{37, 38, 39\} \rightarrow R = 52$;
- Test B.2: N=100, D=2
  - $C_{D1} = \{37, 38\} \rightarrow R = 61$;
  - $C_{D2} = \{39\} \rightarrow R = 11$;

I risultati di questo test risultano piuttosto interessanti:

- Se si confronta il test B.1 con il test B.2 è possibile confermare una netta perdita di pacchetti dovuta al frequency hopping. Si tratta, precisamente, di una differenza di 20 pacchetti che, probabilmente, sarebbe stata maggiore in caso di utilizzo di tre chiavette distinte.
- Il test B.1 risulta addirittura peggiore del test A.1. Si può quindi concludere che la possibilità di ricevere pacchetti su un canale in più rispetto a quanto proposto nel test A.1 amplifica il problema introdotto dal frequency hopping nel test B.1, causando la ricezione di un minore numero di pacchetti.
- Il test B.2 conferma una percentuale simile di pacchetti persi a causa del frequency hopping osservata anche nel test A.1 (per quello che riguarda l'ascolto dei canali 37 e 38 con la stessa chiavetta).
- Basandosi sulla distribuzione dei pacchetti osservata nel test preliminare, si può affermare che il test B.1 è caratterizzato da una perdita di pacchetti pari al 37%, corrispondente a più del doppio rispetto a quanto osservato nel test A.1 indicando una crescita netta dell'impatto del frequency hopping con l'aumentare del numero di canali osservati. Il test B.2, invece, presenta una perdita di pacchetti del solo 13%. Considerando, inoltre, che il numero di pacchetti ricevuti sul canale 39 corrisponde al 100% di quelli identificati dal test preliminare, si può affermare che la totalità dei pacchetti persi sia dovuta al frequency hopping presente sulla chiavetta con C=$\{37,38\}$.

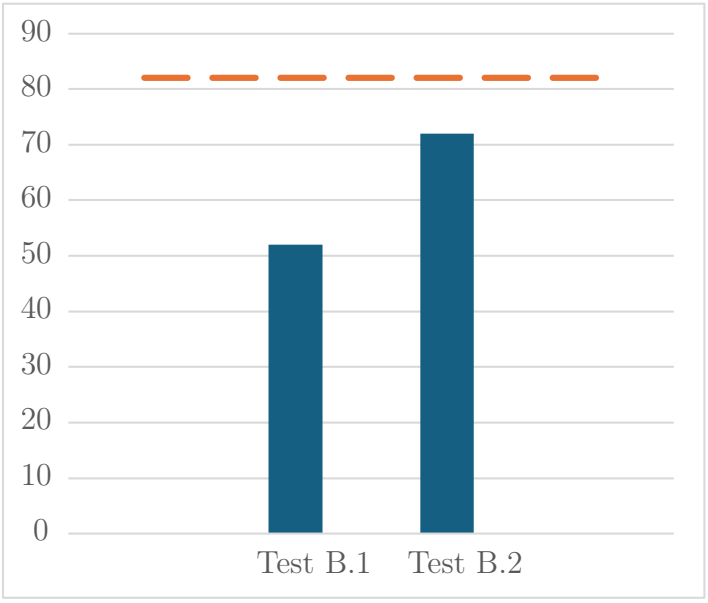

Figura 4.4: Il test B.2 è colpito più lievemente dal frequency hopping in quanto salta tra due soli canali

In conclusione si può affermare, tramite il test preliminare, che il numero di pacchetti persi a causa di interferenze o altri aspetti intrinsechi dell'ambiente di svolgimento dei test ammonta a 18, corrispondente al 18%. Tutti gli altri pacchetti persi sono dovuti invece al meccanismo del frequency hopping, il cui effetto aumenta con l'aumentare del numero di canali da osservare. Il test che meglio rappresenta la realtà è il B.1: un server configurato in modo standard ascolta tutti e tre i canali di advertising ma questo provoca una perdita di pacchetti pari al 48%, combinando la perdita dovuta al frequency hopping e quella dovuta alle interferenze e altri fattori dell'ambiente. Chiaramente, si tratta solo di un risultato sperimentale: un server potrebbe essere in grado di ridurre notevolmente questa percentuale se dotato di un controller apposito in grado di gestire in modo più appropriato il salto tra i tre canali. Inoltre, potrebbe essere implementato un meccanismo lato client che consenta il reinvio automatico di una richiesta di connessione in caso di mancata risposta da parte del server. Questa operazione avverrebbe nel corso di pochi millisecondi, risultando completamente "invisibile" all'utilizzatore finale. I test proposti in questa sezione, infatti, non dimostrano un fattore che porta all'inutilizzabilità del BLE, mirano piuttosto a fornire un'indagine statistica delle conseguenze relative all'impiego del frequency hopping. Si noti, inoltre, che sarebbe stato possibile configurare il server in maniera tale da inviare pacchetti di advertising solo su alcuni dei tre canali riservati e ciò avrebbe sicuramente portato ad una riduzione del fattore di perdita dovuto al frequency hopping. Questo, però, non rappresenta la situazione più diffusa in commercio: si ricordi infatti che, come descritto alla sezione 2.1.6, il frequency hopping è un meccanismo che, al costo dell'introduzione di eventuali perdite di pacchetti, permette di fornire una connessione più stabile consentendo di adattare le frequenze utilizzate in base alle interferenze presenti.

Capitolo 5

Conclusioni

Essendo una tecnologia fondata su compromessi, il BLE non richiede l'utilizzo obbligatorio di protocolli di comunicazione sicuri, come invece è ormai abitudine in altri contesti (ad esempio il Web). In particolare, per far fronte alle diverse esigenze dell'utente finale, il BLE permette di eseguire trasmissioni in modalità connessa e non connessa. Per quest'ultima tipologia di comunicazione la sicurezza non viene garantita in alcun modo. Fra le diverse attività svolte nell'ambito della tesi, l'implementazione proposta relativa al reverse engineering dei pacchetti di advertising di un beacon personalizzato (sezione 3.2) dimostra con successo che, nonostante sia effettivamente possibile definire l'invio di pacchetti in formato non standard, questo non è sufficiente per proteggere i dati trasmessi da eventuali host malevoli presenti in rete. Data l'assenza di crittografia in questa modalità di trasmissione dei dati, un qualsiasi dispositivo malevolo potrebbe ricevere i dati in chiaro e tentare di decodificarli come dimostrato nell'esempio proposto in tabella 3.1. Si precisa che i dati trasmessi nell'esempio della tabella 3.1 sono di dimensioni molto ridotte ma l'analisi svolta, seppur grossolana, può essere rappresentativa di tanti contesti reali. Si conclude, comunque, che l'utilizzo di Manufacturer Specific Data non è adatto ad una trasmissione di dati sensibili.

Per quello che riguarda la modalità di comunicazione orientata alla connessione, invece, lo standard Bluetooth definisce due ulteriori sotto-tipologie: la comunicazione non sicura e la comunicazione tramite pairing. La comunicazione non sicura, come descritto dal nome stesso, non garantisce alcuna sicurezza e la differenza dal metodo di comunicazione non orientato alla connessione risiede nel fatto che viene effettivamente stabilita una connessione tra due host, al contrario della trasmissione tramite advertising che si basa sull'invio di pacchetti in broadcast. Una parte importante della tesi si concentra proprio sulla comunicazione non sicura. Questa scelta è stata dettata dal fatto che molto spesso la comunicazione non sicura viene utilizzata da dispositivi comunemente impiegati nell'industria: rappresentando un buon compromesso tra usabilità e consumo energetico, la comunicazione non sicura viene spesso scelta dai programmatori per la realizzazione del sistema finale. La tipologia di comunicazione orientata alla

connessione, sia nella sua versione non sicura che nella versione con pairing, impiega il meccanismo del frequency hopping. Grazie a questo meccanismo, una trasmissione in corso ha la possibilità di "saltare" tra una frequenza e l'altra in modo da migliorare la stabilità della comunicazione. Entrambi i dispositivi devono però accordarsi sulla channel map da utilizzare (sezione 2.1.6) in modo da sincronizzare i salti tra un canale e l'altro e mantenere attiva la connessione. Si potrebbe quindi pensare che il meccanismo del frequency hopping impedisca a priori ad un eventuale dispositivo malevolo di interferire con la comunicazione: non avendo a disposizione la channel map, infatti, un dispositivo non partecipante alla connessione non dovrebbe riuscire a seguire i salti tra un canale e l'altro. L'attacco Man In The Middle simulato nel progetto di tesi (sezione 3.3) vuole dimostrare come un dispositivo malevolo possa superare l'ostacolo del frequency hopping agendo durante la fase di connessione tra i due dispositivi legittimi. In particolare si è dimostrato come, tramite l'esecuzione dei passaggi riportati a sezione 2.2.4, sia possibile indurre client e server legittimi ad instaurare una connessione con il dispositivo intermediario. Quest'ultimo risulta totalmente invisibile a client e server legittimi che, pur non essendo realmente connessi, percepiscono la comunicazione come diretta. La tesi propone anche un'implementazione funzionante di un attacco di tipo replay (sezione 3.3.6), che può essere eseguito successivamente all'attacco MITM. Si conclude quindi che:

- Nel contesto delle modalità di comunicazione non sicure offerte dal BLE, l'assenza di un meccanismo di crittografia dei dati implica la trasmissione in chiaro dei dati stessi. Sebbene il meccanismo del frequency hopping costituisca effettivamente un ostacolo tecnico all'intercettazione dei dati, non è sufficiente per garantire la confidenzialità dei dati trasmessi.

- L'assenza di un meccanismo di autenticazione permette l'esecuzione dell'attacco MITM: il dispositivo intermediario risulta trasparente a client e server legittimi proprio perché non è garantita l'autenticità dei dati.

- Come dimostrato dall'attacco replay, non è presente alcun meccanismo di segnalazione di errore in caso di ricezione di pacchetti duplicati. Inoltre, l'assenza di un meccanismo di numerazione dei pacchetti, permette la memorizzazione di un pacchetto durante un attacco MITM e la sua successiva replicazione in una connessione futura tramite un attacco replay.

Infine, parte della tesi è dedicata all'analisi dell'effetto del frequency hopping in fase di inizializzazione della connessione. In particolare, tramite i test effettuati e descritti alla sezione 4.2, si dimostra una perdita di pacchetti dovuta al salto sequenziale tra i canali 37,38 e 39 durante la fase di advertising e inizializzazione della connessione. In questo caso, comunque, occorre tenere in considerazione che il meccanismo del frequency hopping, seppur rendendo tecnicamente più complessa la comunicazione, permette

di migliorare la stabilità evitando automaticamente le frequenze che presentano interferenze del segnale. Difatti, eventuali operazioni di ritrasmissione dei pacchetti in caso di smarrimento verrebbero effettuate in pochi millisecondi, rendendo il processo del tutto trasparente all'utilizzatore finale. In conclusione, si può affermare che il meccanismo del frequency hopping non costituisce in alcun modo una garanzia di sicurezza e introduce una possibilità che un pacchetto venga perso, ma è proprio grazie a questo meccanismo che le comunicazioni BLE continuano a funzionare anche in ambienti in cui sono presenti interferenze di segnale. Anche in questo caso, quindi, si tratta di un trade-off tra performance, ridotta a causa dell'eventuale ritrasmissione dei pacchetti, e stabilità.

5.1 Sviluppi futuri

La comunicazione basata su pairing, a sua volta, si suddivide in diverse tipologie che non sono argomento della presente tesi. Il pairing nasce per garantire la sicurezza dei dati trasmessi al costo, ovviamente, di un maggiore dispendio energetico e un peggioramento della performance, entrambi aspetti derivanti dall'introduzione di un overhead dovuto proprio all'esecuzione dei vari meccanismi di sicurezza. Non tutte le tipologie di pairing, però, garantiscono una completa sicurezza. In particolare, il metodo di pairing "Just Work", viene utilizzato come forma di minima garanzia di sicurezza su dispositivi non dotati di uno schermo e/o tastiera. Il metodo Just Work, infatti, prevede che entrambi gli host partecipanti alla comunicazione si autentichino tramite lo scambio di una chiave generata casualmente in fase di connessione: a differenza di tutti gli altri metodi di pairing, la generazione della chiave non richiede alcuna azione da parte dell'utilizzatore del client e/o del server (in quanto l'assenza di schermo e/o tastiera impedisce l'effettiva esecuzione di una qualsiasi azione). Questo fattore rappresenta un potenziale rischio dal punto di vista della sicurezza: seppur trattandosi di un metodo basato su pairing, Just Work è vulnerabile all'esecuzione di un attacco MITM in fase di connessione, esattamente come osservato nella simulazione dell'attacco MITM per una comunicazione non sicura. L'implementazione dell'attacco MITM in caso di connessione effettuata con autenticazione Just Work potrebbe, a tutti gli effetti, rappresentare una possibile estensione al software sviluppato a supporto di questa tesi.

Risulta di particolare interesse anche un possibile sviluppo futuro relativo al frequency hopping. Come descritto a sezione 2.1.7, occorre intercettare e leggere il pacchetto CONNECT_REQ per ottenere tutti i parametri necessari all'esecuzione dell'algoritmo di selezione del canale in modo tale da riuscire a seguire la comunicazione tra due host. Questo passaggio potrebbe però essere evitato in determinate circostanze se si hanno a disposizione i giusti strumenti. Innanzitutto, si specifica che il meccanismo descritto brevemente di seguito si riferisce esclusivamente al Channel Selection Algorithm 1 (CSA#1). Tale algoritmo, utilizzato da standard in comunicazioni in cui almeno uno

dei dispositivi partecipanti abbia una versione del Bluetooth inferiore alla 5.0, è il più semplice dei due esistenti ed è descritto matematicamente dalla formula seguente:

$$ch_k = \mathrm{mod}(ch_{k-1} + h_{\mathrm{inc}}, 37)$$

La formula riportata sopra definisce il canale da utilizzare alla trasmissione k sulla base del canale utilizzato alla trasmissione $k-1$, incrementato di una quantità definita in fase di connessione h_{inc}, pari all'incremento del canale utilizzato tra una trasmissione e l'altra. La caratteristica fondamentale del CSA#1 è data dal fatto che la mappa dei canali utilizzati e calcolati dall'algoritmo si ripete ogni 37 trasmissioni per via dell'operazione modulo, necessaria al fine di garantire che il canale utilizzato per la trasmissione $k-1$ corrisponda ad uno dei canali utilizzabili. Se, per esempio, un canale è stato utilizzato alla trasmissione $k = 5$, allora verrà riutilizzato alla trasmissione $k = 42$. Per riuscire a seguire con successo una comunicazione occorre conoscere la mappa di canali utilizzata c_{map}, il tempo tra un salto e l'altro c_{int}, e l'incremento del canale ad ogni salto h_{inc}. Rimanendo in ascolto su uno stesso canale casuale, è sufficiente ricevere due pacchetti appartenenti alla connessione che si desidera seguire per venire a conoscenza dell'intervallo c_{int}: dato che ogni canale utilizzato viene ripetuto ogni 37 trasmissioni occorre dividere la differenza di tempo tra la ricezione del primo e del secondo pacchetto per 37. Sulla base di c_{int}, è possibile rimanere in ascolto su ognuno dei 37 canali dedicati alla trasmissione dei dati per calcolare i dati mancanti e, cioè, l'incremento h_{inc} e la mappa dei canali utilizzata c_{map}. Una volta ricavati tutti i dati necessari è possibile seguire la comunicazione nonostante non si sia intercettato il pacchetto CONNECT_REQ. Si potrebbe quindi implementare un software su un microcontrollore, ad esempio un ESP32, al fine di invertire l'algoritmo CSA#1. Innanzitutto, il software dovrebbe eseguire una prima fase costituita dall'ascolto di ognuno dei 37 canali, sequenzialmente: per ogni canale è sufficiente ricevere due pacchetti, al secondo pacchetto ricevuto si può ascoltare il canale successivo. Una volta completata questa fase, sulla base del tempo di ricezione di ogni pacchetto, è possibile calcolare tutti i parametri necessari alla replicazione dell'algoritmo di frequency hopping eseguito da client e server, in modo tale da riuscire a intercettare la comunicazione. Si precisa che è possibile che uno stesso canale venga, in realtà, riutilizzato più volte all'interno della stessa sequenza di 37 trasmissioni: in tal caso il procedimento per il calcolo di c_{int} varia lievemente. Per ulteriori dettagli e per una descrizione più rigorosa del problema si invita a fare riferimento al paper originale [7].

Bibliografia

[1] Amazon Web Services. Freertos, an open-source real-time operating system kernel for embedded devices. https://www.freertos.org/.

[2] Bluetooth SIG. Bluetooth core specification 4.2. https://www.bluetooth.com/specifications/specs/core-specification-4-2/, 2014.

[3] Bluetooth SIG. Bluetooth core specification supplement 11. https://www.bluetooth.com/specifications/specs/core-specification-supplement-11/, 01 2023.

[4] Bluetooth SIG. Bluetooth assigned numbers. https://www.bluetooth.com/specifications/assigned-numbers/, 09 2024.

[5] Espressif. Espressif iot development framework. https://github.com/espressif/esp-idf.

[6] Henrik Blidh. A cross platform bluetooth low energy client for python using asyncio. https://github.com/hbldh/bleak.

[7] Julian Karoliny, Thomas Blazek, Andreas Springer, and Hans-Peter Bernhard. Predicting the channel access of bluetooth low energy. In *ICC 2023-IEEE International Conference on Communications*, pages 1756–1761. IEEE, 2023.

[8] Nordic Semiconductor. nrf52840 dongle. https://www.nordicsemi.com/Products/Development-hardware/nRF52840-Dongle.

[9] Scapy. Scapy bluetooth usage documentation. https://scapy.readthedocs.io/en/latest/layers/bluetooth.html.

[10] Scapy. Scapy: the python-based interactive packet manipulation program library. https://github.com/secdev/scapy. Versione 2.5.0.