

ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA
Corso di Laurea in Ingegneria Informatica

**REALIZZAZIONE DI UN DISPOSITIVO
PER IL TEST DELLA SICUREZZA
DI RETI ZIGBEE BASATO SU
FRAMEWORK KILLERBEE E
CHIP CC2531**

Relatore:
Chiar.mo Prof.
MARCO PRANDINI

Presentata da:
MARTINO TOMMASINI

Correlatori:
DAVIDE BERARDI
ANDREA MELIS

I Sessione
Anno Accademico 2019/2020

Introduzione

Negli ultimi anni si è assistito a un rapido sviluppo del mondo dell'Internet of Things e, in prospettiva, la crescita sembra essere ancora più marcata. Data l'ampia diffusione delle nuove tecnologie IoT e la loro costante presenza nella vita quotidiana delle persone, lo studio della sicurezza di questi dispositivi assume un ruolo fondamentale. La tecnologia Zigbee rappresenta una soluzione estremamente popolare tra i protocolli IoT ed è proprio all'interno di questo popolare standard che si colloca lo studio riportato in questo documento.

Due strumenti si distinguono nell'analisi di sicurezza di reti Zigbee: Killerbee, framework professionale di penetration test noto per la gamma di funzionalità implementate e per il suo ampio utilizzo, e il CC2531, dispositivo hardware popolare nel mondo Zigbee per il ridotto prezzo di acquisto rispetto ai prodotti concorrenti disponibili sul mercato.

Alcune soluzioni che permettono la compatibilità tra KillerBee e CC2531 sono attualmente disponibili ma supportano unicamente le funzionalità passive di intercettazione del traffico. Lo scopo di questo studio è quello di realizzare una soluzione basata su KilleBee e CC2531 che permetta di estendere le funzionalità attualmente supportate fornendo la capacità di trasmettere attivamente pacchetti all'interno della rete Zigbee.

Il raggiungimento di questo obiettivo fornirebbe uno strumento estremamente economico e disponibile sul mercato per lo studio completo delle vulnerabilità del protocollo Zigbee. Dato l'esiguo costo del dispositivo, permetterebbe a un crescente numero di persone di partecipare all'analisi di sicurezza e contri-

buire allo sviluppo della comunità Zigbee. Inoltre, renderebbe più accessibile l'acquisto di copie multiple del dispositivo, necessarie per la simulazione di attacchi più complessi.

Indice

Introduzione	1
Elenco delle figure	5
1 Scenario applicativo	7
1.1 Zigbee	7
1.1.1 Protocollo Zigbee	8
1.1.2 Struttura della rete Zigbee	10
1.1.3 Sicurezza	12
1.2 Penetration test in ambienti di reti domotiche	14
1.2.1 Enumerazione	14
1.2.2 Sniffing della chiave	14
1.2.3 Attacchi al Touchlink Commissioning	15
1.2.4 Jamming del segnale	17
1.2.5 Attacco replay	17
2 Strumenti	19
2.1 KillerBee	19
2.1.1 Organizzazione	20
2.1.2 Attacchi forniti	20
2.1.3 Dispositivi hardware compatibili	22
2.2 Altri strumenti	26
2.2.1 Z3sec	26
2.2.2 SecBee	26

2.2.3 Zigdiggity	27
3 Implementazione	29
3.1 Obiettivo	29
3.2 Preparazione dell'ambiente	30
3.3 Ricerca	30
3.3.1 Firmware ZBOSS	31
3.3.2 Considerazioni ricetrasmittente CC2531	32
3.4 Implementazione firmware	32
3.4.1 Ricezione	33
3.4.2 Trasmissione	33
3.5 Implementazione driver	37
3.5.1 Ricezione	37
3.5.2 Trasmissione	39
3.6 Test delle nuove funzionalità	41
3.6.1 Componenti hardware utilizzati	41
3.6.2 Test attacchi KillerBee	41
Conclusioni e sviluppi futuri	47
Bibliografia	49
Ringraziamenti	53

Elenco delle figure

1.1 Architettura Zigbee [4]	8
1.2 Topologie di rete Zigbee [4]	12
2.1 Atmel RZ RAVEN USB Stick.	23
2.2 TelosB mote	23
2.3 River Loop ApiMote	24
2.4 Sewio Open Sniffer	24
2.5 CC2531 Dongle USB	25
3.1 Funzione di invio di pacchetti nel driver KillerBee per CC2531 non implementata.	31
3.2 Implementazione della funzione di trasmissione dati	34
3.3 Implementazione ricezione dati da macchina host	36
3.4 Caricamento driver KillerBee per CC2531 con firmware ZBOSS esteso	37
3.5 Segnalazione inizio procedura di sniffing	38
3.6 Codice driver di lettura dati da CC2531	39
3.7 Codice driver di scrittura dati a CC2531	40
3.8 Test comando zbid	42
3.9 Test comando zbstumbler	43
3.10 Test comando zbwreshark	44
3.11 Test comando zbreplay	45
3.12 Traffico tra CC2531 e lampada IKEA durante attacco replay	46

Capitolo 1

Scenario applicativo

1.1 Zigbee

ZigBee è uno standard wireless prodotto da ZigBee Alliance [\[1\]](#) a partire dal 2002 e sviluppato con lo scopo di realizzare una soluzione universale per il mondo IoT, in cui tutti i dispositivi intelligenti possano collaborare tra di loro, anche se provenienti da produttori diversi. Zigbee Alliance comprende ora le più grandi compagnie mondiali tra cui Apple, Amazon, Philips, IKEA. ZigBee è uno standard aperto utilizzato per la creazione di reti wireless personali affidabili, a bassa potenza e che richiedono una bassa velocità di trasferimento dati. Opera secondo le specifiche IEEE 802.15.4 ed è in grado di coprire una distanza massima di 100 metri.

A seguire sono descritte le principali caratteristiche che hanno reso Zigbee una delle soluzioni maggiormente adottate per le comunicazioni wireless nei dispositivi IoT:

- È più semplice da progettare e meno costosa, se confrontata con le tecnologie Bluetooth e Wifi.
- L'esiguo utilizzo di energia permette di garantire una lunga durata delle batterie.

- Supporta la topologia di rete a maglia (*Mesh Network*), mostrata in Figura 1.2. Ciò permette di aumentare la distanza totale di trasmissione e migliorare l'affidabilità della rete, che è in grado di compensare autonomamente i guasti.
- È semplice da installare e non richiede manutenzione.
- Permette di collegare un ampio numero di dispositivi.

1.1.1 Protocollo Zigbee

Il protocollo Zigbee è strutturato in quattro livelli come mostrato in Figura 1.1. I due livelli inferiori seguono lo standard IEEE 802.15.4 mentre il livello di rete e il livello applicativo sono definiti dalle specifiche Zigbee [2].

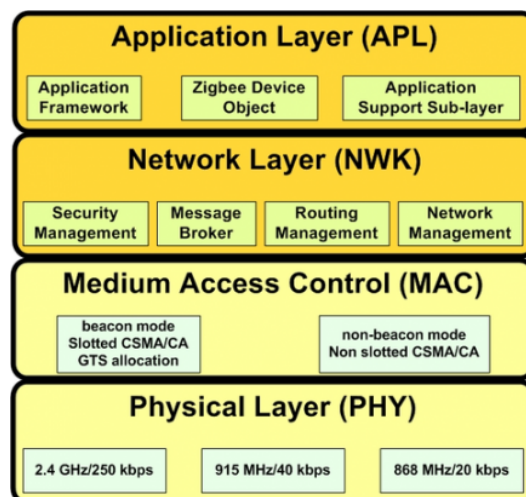


Figura 1.1: Architettura Zigbee [4]

Livello PHY

Il livello fisico esegue la modulazione sui segnali in uscita e la demodulazione sui segnali in arrivo per trasmettere e ricevere bits di informazioni. Si occupa di controllare, accendere, spegnere le antenne radio e selezionare e monitorare i canali di comunicazione. Le frequenze su cui opera lo strato fisico del protocollo Zigbee sono le seguenti: [3]

- 868 MHz, disponibile in Europa
- 915 MHz, disponibile in America e Australia.
- 2,4 GHz, disponibile in tutto il mondo e la più comunemente usata.

I canali e le velocità di trasmissione corrispondenti sono mostrate in Tabella 1.1.

Frequenze	Canali	Velocità di trasmissione
868-868,6 MHz	0	20 Kbps
902-928 MHz	1-10	40 Kbps
2400-2483,5 MHz	11-26	250 Kbps

Tabella 1.1: Frequenze e Canali Zigbee

Livello MAC

Il livello MAC si occupa di definire le politiche per la condivisione della banda da parte dei dispositivi che operano sulle stesse frequenze. I protocolli ZigBee supportano sia reti *non beacon enabled* sia reti *beacon enabled*. Per i primi si utilizza CSMA/CA¹, mentre per i secondi si utilizzano meccanismi basati sia su GTS² che CSMA/CA^[4].

Altre funzionalità del livello MAC sono quelle relative al controllo, all'acknowledgment, all'assemblaggio e alla frammentazione dei frames.

Esistono quattro tipi diversi di frame Zigbee (Beacon, Data, Command e Acknowledge), ciascuno di essi con campi e lunghezze variabili^[3].

Livello NWK

Il livello di rete fornisce servizi riguardanti l'inizializzazione della rete, l'assegnamento degli indirizzi, l'aggiunta o la rimozione di dispositivi, l'instradamento dei messaggi all'interno della rete Zigbee e la trasmissione sicura

¹Carrier Sense Multiple Access / Collision Avoidance

²Guaranteed Time Slot

dei frame [3].

Livello APL

Il livello applicativo rappresenta il livello più in alto dello stack Zigbee. Offre i servizi per interagire con il livello di rete e definisce le applicazioni supportate dal dispositivo.

Il livello APL si compone dei seguenti sotto-livelli: [3]

- **Zigbee Device Object (ZDO):** Permette di definire il ruolo del nodo Zigbee all'interno della rete e le funzionalità necessarie al suo corretto funzionamento. I nodi e i ruoli della rete Zigbee sono trattati nel Paragrafo 1.1.2.
- **Application Support Sub-Layer (APS):** È responsabile del filtraggio dei pacchetti per i dispositivi finali, verifica la duplicità dei pacchetti e permette di automatizzare la ritrasmissione di pacchetti che non hanno ricevuto conferma. Inoltre, si occupa di interagire con la corrispondente applicazione del dispositivo Zigbee destinatario.
- **Application Framework (AF):** Permette di facilitare l'interazione tra le applicazioni e il livello APL.

1.1.2 Struttura della rete Zigbee

Lo standard Zigbee definisce tre diversi tipi di nodo all'interno della rete Zigbee [3]. Ciascuno di essi ha un insieme di funzionalità ben definito:

- **Coordinator (ZC):** Ha il compito di inizializzare la rete Zigbee consentendo ad altri nodi di unirsi a essa. Una volta stabilita la rete, il coordinatore ha un ruolo di routing dei messaggi tra i nodi ed è anche in grado di inviare o ricevere dati. Il ZC deve essere unico all'interno della rete.

- **Router (ZR):** Ha la responsabilità di estendere la rete Zigbee permettendo ad altri dispositivi di unirsi come nodi figli. Assume il compito di instradare pacchetti tra dispositivi ed è presente come nodo intermedio all'interno della rete. La presenza di dispositivi ZR non è obbligatoria.

- **End Device (ZED):** Comunica con il nodo genitore inviando o ricevendo dati ma non ha capacità di routing. Un dispositivo ZED è presente come nodo terminale all'interno della rete Zigbee e non può avere nodi figli.

In Figura [1.2](#) sono mostrate le topologie di rete supportate da Zigbee. La struttura di rete utilizzata varia in base al tipo di dispositivi a disposizione, la distanza tra essi e le funzionalità che si vogliono ottenere. A titolo di esempio, nel caso di strutture di rete complesse che richiedono un livello più alto di affidabilità, la scelta della rete a maglia (*Mesh Network*) è la preferibile.

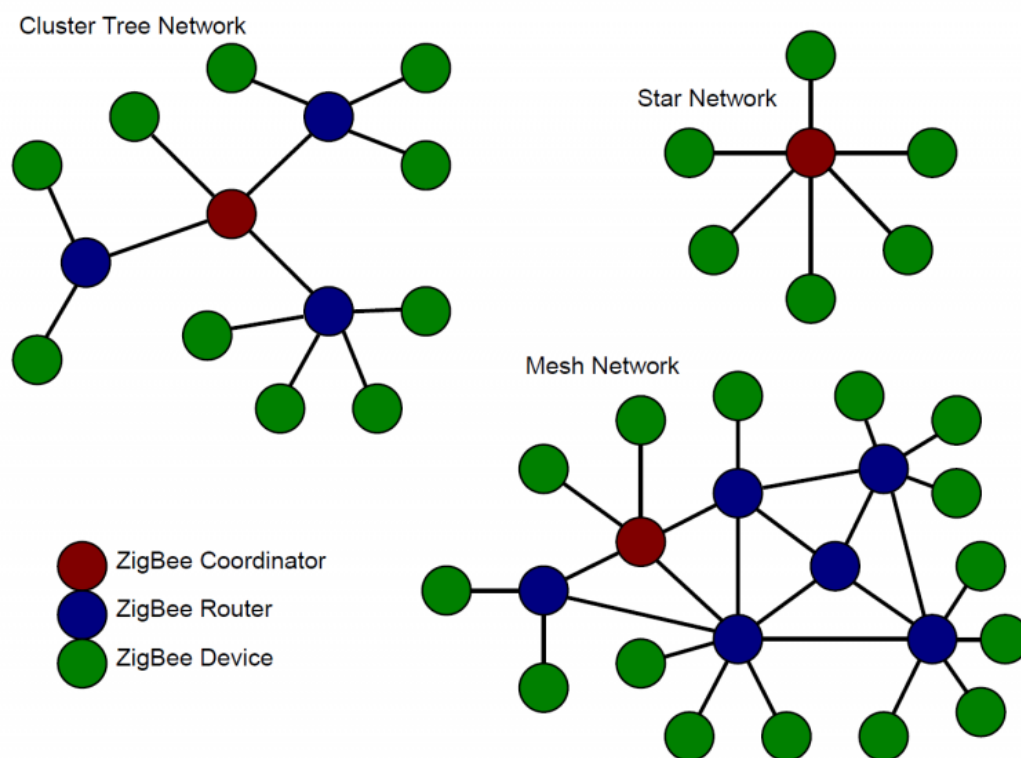


Figura 1.2: Topologie di rete Zigbee [4]

1.1.3 Sicurezza

La riservatezza del traffico Zigbee è garantita dall'utilizzo dell'algoritmo di cifratura AES a 128 bit. La sicurezza del protocollo si basa principalmente sull'utilizzo di due chiavi: [3]

- **Link Key:** Chiave a 128-bit che viene utilizzata per una comunicazione unicast tra pari entità a livello applicativo. La chiave viene solitamente condivisa tra due dispositivi al momento della loro installazione in fabbrica.
- **Network Key:** Chiave a 128-bit che viene utilizzata per le comunicazioni broadcast e per le comunicazioni a livello di rete. La chiave di rete è condivisa tra tutti i dispositivi. La Network Key viene fornita

all'End Device, cifrata utilizzando la Link Key, quando il dispositivo si unisce alla rete Zigbee. Tale chiave sarà utilizzata per cifrare tutti gli scambi successivi di pacchetti, rendendo inefficace il tentativo di intercettazione del traffico da parte di un attaccante.

Avendo la matematica certezza che la cifratura AES a 128 bit sia in grado di mantenere la riservatezza nelle comunicazioni, è necessario assicurarsi che la chiave di cifratura sia distribuita in modo sicuro. Esistono due diversi modelli di sicurezza per la gestione delle chiavi:

- **Modello Centralizzato:** Prevede l'utilizzo di un dispositivo che assume il ruolo di Trust Center (TC), solitamente è il Coordinator della rete Zigbee. Il Trust Center forma una rete centralizzata, configura e autentica i dispositivi. Inoltre, permette di definire un'unica Link Key per ogni dispositivo che si unisce alla rete e trasferisce loro la Network Key in forma cifrata. [5]
È il modello che garantisce una maggiore sicurezza ma è anche il più complesso.
- **Modello Distribuito:** È composto da due tipi di dispositivi: Router e End Device. Un Router può creare una rete di sicurezza distribuita quando non riesce a trovare alcuna rete esistente. Ogni Router si comporta da Trust Center e può emettere la Network Key. Ogni volta che un nuovo nodo si unisce alla rete, i Router preesistenti inviano la Network Key ai nuovi, cifrandola con la Link Key. Per questo motivo è necessario che i dispositivi siano stati preconfigurati con la Link Key. Tutti i dispositivi della rete utilizzano la stessa Network Key. [5]
Fornisce un sistema meno sicuro ma più semplice.

1.2 Penetration test in ambienti di reti domestiche

Dato il vasto utilizzo della tecnologia Zigbee nel mondo IoT, il test di sicurezza di queste reti risulta fondamentale per comprenderne i suoi punti deboli e i suoi punti di forza.

In questa Sezione saranno trattati alcuni tra i principali attacchi a una rete Zigbee per evidenziare le sue vulnerabilità. Gli strumenti che è possibile utilizzare per le analisi qui trattate sono descritti nel Capitolo [2](#).

1.2.1 Enumerazione

Gli attacchi di enumerazione rappresentano il passo iniziale per lo studio delle reti Zigbee. Permettono di ricavare informazioni sulla rete e sui dispositivi al suo interno, fornendo un punto di partenza per gli attacchi successivi. L'attacco consiste nell'invio di un pacchetto sui diversi canali della rete Zigbee, descritti nel Paragrafo [1.1.1](#), richiedendo informazioni riguardanti i dispositivi attivi. La risposta ricevuta permette di ricavare informazioni dettagliate sui componenti, sulle configurazioni della rete, sugli indirizzi dei dispositivi e sul canale Zigbee utilizzato [\[2\]](#).

Il pacchetto di richiesta dati inviato dall'attaccante potrebbe essere sia un *beacon request* che una *scan request*, in base al tipo di dispositivi e rete esistente.

Un esempio concreto di enumerazione basato su *beacon* è mostrato nel Paragrafo [3.6.2](#).

1.2.2 Sniffing della chiave

Dal momento che la segretezza delle chiavi è fondamentale per garantire la sicurezza del protocollo Zigbee, come descritto nel Paragrafo [1.1.3](#), gli attacchi volti a intercettare la distribuzione delle chiavi sono quelli che possono arrecare più danno in caso di successo. Un attaccante che entra in possesso

della Network Key, utilizzata per cifrare i pacchetti, ha controllo totale sui dispositivi all'interno della rete.

È possibile catturare la Network Key in fase di distribuzione della chiave che avviene quando un nuovo nodo si unisce alla rete, vedi Paragrafo [1.1.3](#). Poiché tale chiave è precedentemente cifrata utilizzando la Link Key, la riservatezza della Network Key deriva dalla segretezza della Link Key. Alcune Link Keys sono state pubblicate su piattaforme online e sono utilizzabili per la decifrazione della Network Key e, di conseguenza, del traffico Zigbee [\[8\]](#), [\[9\]](#). Inoltre, nel caso si è in possesso del dispositivo hardware utilizzato, è possibile identificare la Link Key attraverso studi di *reverse engineering* [\[9\]](#).

Per aumentare la finestra di tempo critica riguardante la fase di scambio di chiavi, l'attaccante può provocare il malfunzionamento di un dispositivo e indurre il proprietario a effettuare una procedura di ritorno alle condizioni di fabbrica. In questo modo, la Network Key viene nuovamente distribuita. Per rendere inefficace questo tentativo, alcuni dispositivi Zigbee mantengono installate le Network Key in memoria.

1.2.3 Attacchi al Touchlink Commissioning

Sono attacchi che permettono di sfruttare le vulnerabilità dei dispositivi Zigbee che utilizzano il *Touchlink Commissioning* [\[3\]](#), ovvero una procedura che definisce la fase di creazione della rete Zigbee e di aggiunta di nuovi nodi a essa. Sono attuabili senza la conoscenza della Network Key perché il traffico non è cifrato durante questa fase e possono essere utilizzati con lo scopo di causare malfunzionamenti della rete. Affinché tali attacchi siano possibili, l'attaccante deve essere a stretto contatto con i dispositivi vittima.

- **Identify Action:** Questo attacco sfrutta la caratteristica della procedura di Touchlink Commissioning che offre la possibilità agli utenti di richiedere ai dispositivi Zigbee di identificarsi, attraverso l'esecuzione di un comportamento predefinito come l'emissione di un suono, l'accensione

di qualche LED, l'intermittenza della luce etc...

La richiesta di identificazione contiene al suo interno l'intervallo di tempo per cui il dispositivo ricevente si deve identificare, rappresentato da due bytes di informazione. Un attaccante può trarre vantaggio da questa modalità e generare un pacchetto *identify request* da trasmettere al dispositivo vittima specificando un esteso intervallo di tempo (il valore massimo specificabile è pari a 18 ore 12 minuti 14 secondi). Molti dispositivi Zigbee, specialmente lampadine, non sono controllabili mentre si identificano, costringendo il proprietario a disconnettere fisicamente la lampadina.

Inoltre, nel caso in cui il proprietario decidesse di resettare il dispositivo alle condizioni di fabbrica, l'attaccante potrebbe approfittare della situazione per intercettare i pacchetti relativi alla fase critica di distribuzione della Network Key, come descritto nel Paragrafo [1.2.2](#)

In ogni caso, la procedura di identificazione porta a un maggiore consumo della batteria per i dispositivi Zigbee che la utilizzano. [\[12\]](#)

- **Reset to Factory-New:** Questo attacco sfrutta la caratteristica della procedura di Touchlink Commissioning che offre la possibilità agli utenti di riportare il proprio dispositivo alle condizioni di fabbrica.

Un attaccante può approfittare di questa funzionalità, costruendo un pacchetto *reset to factory new request* e trasmettendolo al dispositivo vittima, causando l'azzeramento di tutte le configurazioni memorizzate nel dispositivo. A seguito di un attacco terminato correttamente, il dispositivo vittima non è più in grado di comunicare con la rete Zigbee ed è necessario un intervento manuale del proprietario per avviare una nuova procedura di *commissioning*. In questa fase l'attaccante potrebbe mettersi in ascolto con lo scopo di intercettare i pacchetti relativi alla fase critica di distribuzione della Network Key, come descritto nel Paragrafo [1.2.2](#). [\[12\]](#)

1.2.4 Jamming del segnale

Dal momento che Zigbee è una tecnologia concepita per garantire comunicazioni a bassa potenza, è particolarmente vulnerabile al rumore presente nel canale di comunicazione usato.

Un attaccante, provvisto di hardware necessario, può emettere onde radio sulle frequenze usate dalla rete Zigbee per interferire e bloccare le comunicazioni tra i dispositivi. L'attacco continua fino a che il proprietario del dispositivo, accortosi del mancato funzionamento dei suoi dispositivi, avvia una nuova procedura di *commissioning*. L'attaccante può quindi mettersi in ascolto e provare a intercettare la Network Key, come descritto nel Paragrafo

[1.2.3](#). [\[7\]](#)

L'attacco è tanto più efficace quanto l'attaccante è posizionato vicino al dispositivo vittima.

1.2.5 Attacco replay

L'attacco replay si basa sulla ritrasmissione all'interno della rete Zigbee di traffico precedentemente catturato, come se fosse il mittente originale a inviare nuovamente i pacchetti. L'effetto di un attacco replay dipende dal contenuto dei pacchetti ritrasmessi.

Nel Paragrafo [3.6.2](#) è possibile trovare un esempio concreto di attacco replay, insieme a una descrizione delle politiche adottate da Zigbee per mitigare questo attacco.

Capitolo 2

Strumenti

In questo capitolo saranno esaminati i principali software utilizzati per lo studio e la compromissione della sicurezza del protocollo Zigbee. Dato il suo forte utilizzo nel campo, particolare attenzione sarà rivolta al framework KillerBee e ai dispositivi hardware con esso compatibili.

2.1 KillerBee

KillerBee [10] è un framework open-source utilizzato per valutare la sicurezza dei sistemi IEEE 802.15.4 e ZigBee. È scritto interamente in Python e dispone di una comunità attiva e collaborativa. Il software, testato e ampiamente usato su piattaforme Linux, è progettato per semplificare il processo di intercettazione dei pacchetti in transito, manipolazione del traffico, decodifica e iniezione di pacchetti. Utilizzando gli strumenti forniti da KillerBee e un'interfaccia radio IEEE 802.15.4 compatibile, è possibile intercettare pacchetti in transito su reti ZigBee, riprodurre il traffico, attaccare i sistemi crittografici e molto altro.

Come espresso dai creatori del framework [10], KillerBee è un software professionale concepito per sviluppatori e analisti con conoscenze del linguaggio Python e del funzionamento del protocollo Zigbee.

2.1.1 Organizzazione

KillerBee è organizzato in moduli che rendono la logica di implementazione degli attacchi indipendente dai dispositivi hardware utilizzati.

A seguire è riportata la struttura del framework con una breve descrizione dei rispettivi contenuti:

- **doc:** documentazione della libreria KillerBee;
- **tools:** strumenti di analisi e di attacco offerti dal framework. Implementano gli attacchi a reti Zigbee e IEEE 802.15.4 astruendo dai dispositivi fisici utilizzati.
- **killerbee:** libreria di KillerBee contenente i driver per i dispositivi supportati.
- **firmware:** firmware disponibili per dispositivi hardware supportati da KillerBee.
- **scripts:** raccolta di programmi vari usati durante lo sviluppo.
- **sample:** alcuni esempi di pacchetti intercettati e salvati in vari formati come pcap e Daintree DCF, utili per testare il funzionamento di alcuni attacchi e analizzare in chiaro un esempio di comunicazione tra dispositivi Zigbee.

2.1.2 Attacchi forniti

A seguire sono descritti i principali strumenti di analisi e di attacco messi a disposizione da KillerBee, ciascuno di essi opportunamente invocabile con il parametro -h per avere informazioni sul loro funzionamento:

zbid: individua i dispositivi disponibili sulla macchina host, compatibili con KillerBee. Particolarmente utile nel caso si utilizzino molteplici dispositivi poiché permette di enumerare e distinguere le interfacce a disposizione.

zbwreshark: permette di catturare i frame IEEE 802.15.4 e di visualizzarli in tempo reale utilizzando Wireshark. È possibile inserire le chiavi di decifrazione in Wireshark per cercare di intercettare la Network Key come descritto nel Paragrafo [1.2.2](#).

zbdump: permette di catturare i frame IEEE 802.15.4 salvandoli in file pcap o Daintree DCF.

zbstumbler: esplora l'area circostante cercando reti Zigbee e IEEE 802.15.4, trasmettendo pacchetti *beacon request* su un particolare canale Zigbee e aspettando un pacchetto *beacon* in risposta. Nel caso tale pacchetto sia ricevuto, alcune informazioni utili sulle reti Zigbee contenute al suo interno sono mostrate a video. Il procedimento è ripetuto per ogni canale Zigbee. I dati visualizzati possono fornire un punto di partenza per procedere con successivi attacchi.

zbreplay: implementa l'attacco replay. Legge i pacchetti da un file pcap o Daintree DCF specificato come parametro e li ritrasmette sull'interfaccia specificata.

zborphanotify: testa il comportamento del Coordinatore della rete Zigbee inviando un *orphan notification frame*, fingendosi un dispositivo all'interno della rete. Tale pacchetto viene solitamente mandato quando un endpoint Zigbee ha perso il contatto con il proprio nodo genitore (Router o Coordinatore).

zbpandconflictlood: Sono necessarie due dispositivi KillerBee per compiere questo attacco. Un'interfaccia KillerBee cattura i pacchetti e contrassegna il loro PAN ID. L'altra interfaccia invia costantemente pacchetti beacon con PAN ID trovati. I pacchetti beacon con lo stesso PAN ID inducono il coordinatore PAN a credere che si sia verificato un conflitto con un'altra rete Zigbee e inizia il processo di riallineamento della rete su un nuovo PAN ID. Il processo si ripete continuamen-

te. Solitamente, i dispositivi di rete non riescono a sostenere il rapido cambiamento e dopo alcuni secondi la rete cade.

zbconvert: converte un file pcap in Daintree DCF e viceversa.

zbscapy: fornisce una shell interattiva per operare con la libreria Scapy utilizzando una interfaccia Killerbee.

zbdsniff: cattura traffico Zigbee alla ricerca della Network Key. La chiave viene stampata appena trovata.

zbassocflood: trasmette continuamente pacchetti di associazione a un dispositivo Zigbee destinatario nel tentativo di provocare un suo malfunzionamento, qualora il target non riesca a gestire l'elevato numero di richieste di associazione.

2.1.3 Dispositivi hardware compatibili

Per poter utilizzare il framework, è necessario disporsi di dispositivi hardware compatibili. I dispositivi radio supportati da KillerBee sono molteplici ma non tutti permettono di sfruttare le funzionalità offerte dal framework. Alcune scelte sono più o meno costose e altre più o meno reperibili sul mercato.

A seguire sono mostrati i principali dispositivi compatibili, specificando le funzionalità che sono supportate, i costi e la relativa disponibilità sul mercato. Una lista completa dei dispositivi può essere trovata sulla pagina Github di KillerBee [\[10\]](#)

– Atmel RZ RAVEN USB Stick

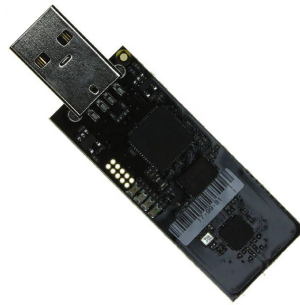


Figura 2.1: Atmel RZ RAVEN USB Stick.

La Atmel RZ RAVEN stick [\[15\]](#) è una chiavetta USB provvista di firmware open source in grado di supportare nativamente le funzionalità di sniffing del traffico Zigbee. Utilizzando il firmware fornito da KillerBee, il dispositivo è compatibile con tutte le funzionalità fornite dal framework [\[10\]](#).

Il prezzo è di circa 35/40 euro ma la sua produzione è molto limitata ed è estremamente difficile reperirla nel mercato.

– TelosB mote



Figura 2.2: TelosB mote

TelosB mote [\[16\]](#) è un sensore wireless con software open source conforme allo standard IEEE 802.15.4, realizzato da MEMSIC. Se utilizzato con il firmware provvisto da KillerBee permette di usare tutte le funzionalità definite dal framework. Il prezzo di mercato si avvicina a 90 euro ed è facilmente acquistabile da vari rivenditori elettronici online.

– River Loop ApiMote

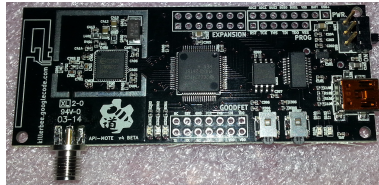


Figura 2.3: River Loop ApiMote

ApiMote [17] è un dispositivo progettato e sviluppato da River Loop Security, nato per studiare le vulnerabilità delle reti Zigbee. Il progetto è open-source e l'hardware è venduto con il firmware di KillerBee già installato. Supporta tutte le funzionalità implementate da KillerBee e il suo costo di mercato è di circa 150 euro.

– Sewio Open Sniffer



Figura 2.4: Sewio Open Sniffer

Open Sniffer [18] è un prodotto realizzato da Sewio, concepito per lo studio dei protocolli IoT. In particolare analizza reti compatibili con gli standard IEEE 802.15.4, Zigbee, 6LoWPAN, Wireless Hart e ISA100.11a. Il dispositivo è compatibile con KillerBee ma è supportato unicamente per le funzionalità relative alla intercettazione e studio di pacchetti in transito. Il

costo del prodotto è di circa 300 euro, acquistabile sul sito ufficiale di Sewio [18].

– CC2531 Dongle USB



Figura 2.5: CC2531 Dongle USB

Il CC2531 Dongle Usb [19] è un ricetrasmittitore IEEE 802.15.4/Zigbee prodotto dalla Texas Instruments e fornito di una interfaccia USB, collegabile a un dispositivo host. Questo prodotto viene venduto preinstallando il firmware proprietario della Texas Instruments [22] che permette di utilizzare la chiavetta come *packet sniffer*.

Il dispositivo è supportato da KillerBee per le funzionalità di intercettazione dei pacchetti e studio di essi attraverso l'interfaccia grafica fornita da Wireshark.

La chiavetta è comunemente disponibile sul mercato e acquistabile dal sito ufficiale della Texas Instruments [19] o da altri rivenditori elettronici come Amazon [20] e AliExpress [21] a un prezzo ridotto che varia dai 5 ai 12 euro. Inoltre, sostituendo il firmware originario e installando il firmware proprietario Z-Stack [24] della Texas Instrumental, è possibile utilizzare la chiavetta CC2531 per sostituire la maggior parte dei gateway ZigBee proprietari e per controllare una varietà di dispositivi diversi, come lampade, sensori di temperatura, interruttori, ecc. [23]

2.2 Altri strumenti

A seguire sono menzionati ulteriori framework che è possibile utilizzare per l'analisi delle vulnerabilità del protocollo ZigBee.

2.2.1 Z3sec

Z3sec [11] è un framework open source di penetrazion testing che comprende una serie di strumenti per lo studio di sicurezza di ZigBee. In particolare, il framework è stato concepito per testare le vulnerabilità dei dispositivi Zigbee che implementano lo standard ZigBee Light Link (ZLL) o lo standard ZigBee 3.0.

Z3sec può inviare comandi touchlink per sfruttare i punti deboli del *touchlink commissioning* [12]. Utilizzando questo software, un attaccante in prossimità dei dispositivi, è in grado di eseguire attacchi descritti nel Paragrafo 1.2.3 o di assumere il controllo dei dispositivi ZigBee. Poiché il traffico durante il *touchlink commissioning* non è cifrato, la conoscenza della *network key* non è fondamentale per il positivo conseguimento degli attacchi forniti. Il software non è più mantenuto dagli sviluppatori e si poggia sul framework KillerBee.

2.2.2 SecBee

SecBee [13] è uno strumento di penetration testing di reti Zigbee sviluppato da Cognosec e destinato a sviluppatori e ricercatori di sicurezza. Fornisce strumenti per testare e sfruttare alcune deboli politiche di sicurezza implementate dal protocollo Zigbee [7].

Il software non è più mantenuto dagli sviluppatori e si poggia sul framework KillerBee.

2.2.3 Zigdiggity

Zigdiggity [14] è un software open source di penetration testing creato da Matt Gleason e Francis Brown di Bishop Fox. L'obiettivo principale di questo framework è attaccare i lucchetti e le serrature intelligenti che implementano il protocollo Zigbee.

Zigdiggity permette di scansionare la rete circostante alla ricerca di dispositivi Zigbee che funzionano come serrature. Una volta trovati è possibile prendere il controllo del dispositivo attraverso un attacco basato sul conflitto di PAN ID nella rete.

Il framework è unicamente compatibile con Raspberry Pi 3+ provvisto di un'interfaccia Raspbee.

Capitolo 3

Implementazione

In questo capitolo verrà descritto il processo che ha portato all'estensione delle funzionalità del framework KillerBee e della chiavetta con chip CC2531. Infine, sarà creato un ambiente per testare le nuove capacità della chiavetta, servendosi di dispositivi Zigbee.

3.1 Obiettivo

Lo scopo principale che ha mosso la ricerca è stato quello di fornire uno strumento compatibile con il framework KillerBee, a basso costo e facilmente reperibile sul mercato, per studiare e attaccare le vulnerabilità delle reti Zigbee.

Tra gli strumenti hardware supportati da KillerBee, vedi Paragrafo [2.1.3](#), il CC2531 rappresenta una scelta estremamente economica e disponibile su gran parte dei rivenditori elettronici online. Tuttavia, il dispositivo è compatibile con KillerBee unicamente per le funzionalità relative allo sniffing del traffico [\[10\]](#).

L'obiettivo è quello di estendere la compatibilità tra KillerBee e CC2531 per rendere utilizzabili tutti gli attacchi forniti dal framework, elencati nel Paragrafo [2.1.2](#).

3.2 Preparazione dell'ambiente

Il framework KillerBee è stato predisposto seguendo i passi descritti dalla pagina ufficiale [10], su una macchina host Linux con distribuzione Ubuntu 18.04 LTS.

Il firmware installato sul CC2531 è il firmware originale della Texas Instruments [22] con cui il dispositivo viene venduto, rendendo possibile l'utilizzo del CC2531 come *packet sniffer*. È possibile installare il firmware servendosi di un CC Debugger [25] e seguendo le istruzioni riportate sul sito Zigbee2mqtt [26]. Esistono soluzioni alternative, in assenza di CC debugger come l'utilizzo di un Raspberry Pi [27] o di una piattaforma Arduino [28]. Guide in merito possono essere trovate sul sito Zigbee2mqtt [26].

3.3 Ricerca

Il passo iniziale risultava capire quale fossero i componenti mancanti affinché la trasmissione di pacchetti fosse possibile. In particolare, capire se mancasse solamente il driver di KillerBee per l'invio di frame utilizzando il CC2531 oppure se fosse il firmware stesso del dispositivo a non fornire alcuna funzionalità di trasmissione di dati.

Attraverso lo studio del framework è stato possibile notare che tutti i driver dei dispositivi hardware supportati sono organizzati in un unico modulo chiamato *killerbee*, come descritto nel Paragrafo 2.1.1. Identificato il file relativo alla compatibilità con il CC2531 è stato possibile analizzare il suo contenuto e dimostrare l'effettiva mancanza di driver per la trasmissione di pacchetti, come testimoniato dalla Figura 3.1.

```
179     def inject(self, packet, channel=None, count=1, delay=0, page=0):
180         """
181         Injects the specified packet contents.
182         @type packet: String
183         @param packet: Packet contents to transmit, without FCS.
184         @type channel: Integer
185         @param channel: Sets the channel, optional
186         @type page: Integer
187         @param page: Sets the subghz page, not supported on this device
188         @type count: Integer
189         @param count: Transmits a specified number of frames, def=1
190         @type delay: Float
191         @param delay: Delay between each frame, def=1
192         @rtype: None
193         """
194         raise Exception('Not yet implemented')
```

Figura 3.1: Funzione di invio di pacchetti nel driver KillerBee per CC2531 non implementata.

Inoltre, come descritto dal sito ufficiale della Texas Instruments [22], il firmware originario del CC2531 non implementa le funzionalità di trasmissione di dati e non è aperto al pubblico. Pertanto, mancando le condizioni necessarie per poter ampliare il firmware con nuove funzionalità, è stato necessario ricercare un nuovo firmware che permettesse di attuare delle modifiche.

3.3.1 Firmware ZBOSS

ZBOSS [29, 30] è uno stack open source certificato da Zigbee Alliance [1] che implementa le specifiche del protocollo Zigbee. È una soluzione multi-piattaforma ad alte prestazioni e memoria ridotta. Fornisce, inoltre, il codice sorgente del progetto ZBOSS sniffer [30], uno sniffer di pacchetti ZigBee multiplatforma, compatibile con l'hardware SmartRf05EB della Texas Instruments e la chiavetta CC2531.

Tale software permette l'utilizzo del CC2531 per l'intercettazione dei pacchetti in transito nella rete Zigbee e la loro visualizzazione attraverso l'interfaccia grafica di Wireshark. Le funzionalità sono comparabili a quelle fornite dal firmware originale della Texas Instruments [22] ma il firmware non è suppor-

tato dal framework KillerBee.

Tuttavia, poiché si ha a disposizione il codice sorgente del firmware, è possibile utilizzarlo per estendere le sue funzionalità e aggiungere la capacità attiva di trasmissione di pacchetti .

Allo stesso modo, poiché KillerBee è un framework open source, è possibile implementare i driver supportare il nuovo firmware esteso del CC2531.

3.3.2 Considerazioni ricetrasmittente CC2531

Prima di passare all'effettiva implementazione delle nuove funzionalità, per evitare lavoro inutile, ci si è chiesto se il ricetrasmittitore del CC2531 sia effettivamente in grado di inviare pacchetti. Due fattori hanno contribuito a chiarificare il dubbio:

- Il datasheet del CC2531 [31] non pone alcuna particolare limitazione riguardante la trasmissione di dati.
- Il CC2531 può potenzialmente essere utilizzato come Coordinatore o Router della rete Zigbee, come menzionato nel Paragrafo 2.1.3. Affinché il CC2531 possa assumere tale ruolo all'interno della rete Zigbee, l'hardware deve necessariamente essere in grado di inviare pacchetti.

3.4 Implementazione firmware

L'ambiente di sviluppo utilizzato per l'estensione del firmware ZBOSS è stato IAR Embedded Workbench per processori 8051 [32], su una macchina host con sistema operativo Windows 10. In una prima fase ci si è servito di un Raspberry Pi 3 per le installazioni del firmware sul dispositivo [26], per poi passare all'utilizzo del CC Debugger [25] che, se configurato con l'IDE IAR , permette l'installazione del firmware al momento della compilazione e fornisce strumenti per il debugging del codice.

Le modifiche apportate al firmware sono a seguire distinte in due aree, una

riguardante la ricezione di dati e una riguardante la trasmissione.

3.4.1 Ricezione

Non è stato necessario apportare molti cambiamenti per quanto riguarda la ricezione di dati, in quanto il firmware packet sniffer di ZBOSS implementa tali caratteristiche nativamente. Non tutte le funzionalità sono però compatibili con quelle che il framework KillerBee si aspetta ed è stato necessario operare qualche cambiamento.

In particolare l'operazione di impostazione del canale e il comando che segna l'inizio della procedura di sniffing sono stati resi indipendenti, in accordo con la logica adottata dal framework KillerBee. Inoltre, tale operazione risulta necessaria poiché attraverso questa funzione sarà possibile impostare il canale su cui trasmettere i dati.

Ulteriori modifiche sono state apportate all' *interrupt handler* dell'antenna ricetrasmittente per garantire la compresenza di trasmissione e ricezione.

3.4.2 Trasmissione

Il firmware packet sniffer di ZBOSS [30] non dispone di alcuna funzionalità riguardante la trasmissione di dati. Facendo tuttavia riferimento al firmware ZBOSSv1.0 [30] che implementa lo stack Zigbee, è verosimilmente possibile trovare una funzione a basso livello che si occupa di trasmettere i frame all'interno della rete Zigbee.

Il codice sorgente è diviso in moduli, ciascuno di essi riguardante un livello diverso dello stack Zigbee, descritto nel Paragrafo 1.1.1. Nello studio di tale firmware, si è prestato maggiore attenzione al livello MAC del protocollo, responsabile della gestione dei frame.

Una volta trovata la funzione ricercata e studiato il suo contenuto, è stato possibile utilizzarla come esempio per implementare la trasmissione nel firmware packet sniffer di ZBOSS.

Il datasheet del chip CC2531 [31] ha fornito un aiuto fondamentale per assegnare i valori ai registri della CPU e per il corretto popolamento della coda di trasmissione. In Figura 3.2 è possibile osservare la funzione che si occupa di scrivere i dati sulla coda di trasmissione dell'antenna radio.

```
void zb_rf_start_tx(){
    /* prepare transceiver */
    /* disable rx interrupt */
    RFIRQM0 &= ~(1<<6);
    /* disable general RF interrupts */
    IEN2 &= ~(0x01);

    /* Flush TXFIFO buffer */
    ZB_RF_ISFLUSHTX();
    /* Enable TX transceiver */
    ZB_RF_ISTXON();
    /* tx done interrupt cleared */
    RFIRQF1 = ~0x02;
    {
        zb_uint8_t i;
        zb_sniffer_transport_rb_record_t *record;

        /* read record from ring buffer */
        record = ZB_RING_BUFFER_GET(&transport_rb);
        /*pointer to the packet to transmit */
        zb_uint8_t *ptr = record->data;
        /* debug: turn red led on */
        ZB_SNIFFER_RED_LED_SET();

        /*record->header.len contains the len of:
        LEN+ MHR + MCD(payload of mac frame).
        FCS is trasmitted automatically (not in ptr) */
        for (i = 0; i<record->header.len; i++)
        {
            RFD = ptr[i];
        }

        /* signal ring buffer that packet has been read */
        ZB_RING_BUFFER_FLUSH_GET(&transport_rb);
    }
    IEN2 |= 0x01; /* enable rf general interrupt back */
    RFIRQM1 |= 0x03;
    RFIRQM0 |= (1<<6);
}
```

Figura 3.2: Implementazione della funzione di trasmissione dati

Come è possibile notare dalla Figura [3.2](#), i dati da trasmettere vengono prelevati dal buffer circolare *transport_rb* e scritti sulla coda di trasmissione attraverso il registro RFD. Si è scelto di utilizzare lo stesso buffer circolare che viene usato in fase di ricezione per limitare il numero di risorse utilizzate, essendo limitato lo spazio di memorizzazione del firmware. Per evitare di avere pacchetti relativi alla trasmissione quando si è in fase di ricezione e viceversa, tale buffer circolare è svuotato a ogni passaggio di stato. I cambiamenti di stato avvengono a seguito di comandi inviati dal driver. Inoltre, per permettere all'utilizzatore della chiavetta di avere un riscontro visivo dell'effettiva trasmissione del frame, viene acceso un LED rosso il momento precedente l'invio di dati e, se il frame è stato trasmesso correttamente, il LED sarà spento all'interno dell'*interrupt handler*, garantendo il successo dell'operazione.

Il buffer circolare *transport_rb* che viene utilizzato per la temporanea memorizzazione dei frame da inviare, viene popolato con i dati provenienti dalla macchina host in cui il CC2531 è collegato. La comunicazione host-chiavetta, già precedentemente implementata per garantire la trasmissione dei pacchetti intercettati da CC2531 al dispositivo host e l'invio di comandi di gestione da host a CC2531, è stata modificata per supportare il passaggio di interi frames da host a CC2531, vedi Figura [3.3](#).

Come osservabile dalla Figura [3.3](#), i dati inviati dall'host e relativi a un comando di gestione (inizia sniffing, ferma sniffing, invia pacchetto, cambia canale) sono memorizzati in un secondo buffer circolare chiamato *cmd_rb* che contiene i comandi prossimi a essere eseguiti. Invece, i dati relativi ai pacchetti da trasmettere sulla rete Zigbee, sono accumulati temporaneamente fino alla completa ricezione di un intero frame. Solo a questo punto viene riempito il buffer circolare del trasporto *transport_rb* che sarà usato per popolare la coda di trasmissione, come mostrato in Figura [3.2](#) e descritto precedentemente in questo paragrafo. L'operazione di accumulo descritta è necessaria poiché la dimensione massima di un pacchetto Zigbee è 127 bytes,

mentre il numero limite di bytes attualmente trasferibili tra host e chiavetta è 64 bytes.

```

USBFW_SELECT_ENDPOINT(4);

if (USBFW_OUT_ENDPOINT_DISARMED() ) {

    // Get length of USB packet, this operation must not be interrupted.
    length = USBFW_GET_OUT_ENDPOINT_COUNT_LOW();
    length += USBFW_GET_OUT_ENDPOINT_COUNT_HIGH() >> 8;
    if (length)
    {
        uint8 i;

        usbfwReadFifo(&USBF4, length, buffer+bytes_arrived);

        // command arrived
        if ( !packet_is_arriving )
        {
            for (i = 0; i < length; i++)
            {
                if (!ZB_RING_BUFFER_IS_FULL(&cmd_rb))
                {
                    ZB_RING_BUFFER_PUT(&cmd_rb, buffer[i]);
                }
            }
        }
        // packet arrived
        // accumulate until we have an entire frame.
        else if (packet_is_arriving && length == USB_MAX_PACKET_SIZE)
        {
            bytes_arrived += length;
        }
        // entire packet arrived. Add in transport ring buffer.
        else if (packet_is_arriving && length < USB_MAX_PACKET_SIZE)
        {
            bytes_arrived += length;
            populate_transport_rb_from_buffer();

            bytes_arrived=0;
            packet_is_arriving = ZB_FALSE;
        }

        USBFW_SELECT_ENDPOINT(4);
        USBFW_ARM_OUT_ENDPOINT();
    }
}

```

Figura 3.3: Implementazione ricezione dati da macchina host

L'utilizzo della USB Firmware Library disponibile sul sito della Texas

Instruments [19], ha permesso di astrarre dai registri della CPU e lavorare a più alto livello per quanto riguarda l'interazione host-CC2531.

3.5 Implementazione driver

Avendo implementato il firmware, è ora necessario scrivere i driver di KillerBee per permettere al framework di essere compatibile con il chip CC2531. Come descritto nel Paragrafo 2.1.1, i driver di ciascun dispositivo sono contenuti all'interno del modulo *killerbee*. L'aggiunta di un nuovo driver KillerBee avviene inserendo un nuovo file Python al cui interno sono implementate le funzioni che permettono di interagire con il dispositivo hardware.

Il framework KillerBee si occupa di caricare i driver necessari al dispositivo USB corrente servendosi del codice prodotto e del codice produttore, definiti nell' *USB Descriptor* all'interno del firmware. La Figura 3.4, mostra la fase di caricamento nel nuovo driver che permetterà di offrire compatibilità con il firmware ZBOSS esteso, descritto nel Paragrafo 3.4

```
elif self.__device_is(CC2531_USB_VEND_ID, CC2531_USB_ZBOSS_PROD_ID):  
    from dev_cc2531_zboss import CC2531  
    self.driver = CC2531(self.dev, self.__bus)
```

Figura 3.4: Caricamento driver KillerBee per CC2531 con firmware ZBOSS esteso

3.5.1 Ricezione

In questo Paragrafo viene descritta l'implementazione del driver KillerBee per permettere al framework di utilizzare il CC2531 come packet sniffer.

In accordo con la logica del framework KillerBee è stato necessario implementare le funzioni che segnalassero al CC2531 l'avvio della procedura di sniffing, la terminazione e l'impostazione del canale Zigbee su cui mettersi in ascolto. Il driver si occupa, inoltre, di lanciare le eccezioni nel caso in

cui il firmware non sia provvisto di particolari funzionalità. In Figura 3.5, è mostrata la funzione che segnala al firmware di iniziare l'intercettazione dei pacchetti sulla rete Zigbee, nel canale specificato come parametro o precedentemente impostato.

```
def sniffer_on(self, channel=None, page=0):
    """
    Turns the sniffer on such that pnext() will start returning observed data.
    Will set the command mode to Air Capture if it is not already set.
    @type channel: Integer
    @param channel: Sets the channel, optional
    @type page: Integer
    @param page: Sets the subghz page, not supported on this device
    @rtype: None
    """
    self.capabilities.require(KBCapabilities.SNIFF)

    if channel is not None:
        self.set_channel(channel, page)

    # Start capture
    self.dev.write(CC2531.USB_EP4_OUT, _to_array(CC2531.USB_START_SNIFFING))
    self.is_sniffing = True
```

Figura 3.5: Segnalazione inizio procedura di sniffing

Inoltre, il framework deve essere istruito su come recuperare e gestire i flussi di byte che sono presentati dal dispositivo hardware sui propri endpoints, una volta che la procedura di sniffing del traffico Zigbee è iniziata. Poiché la dimensione massima di un pacchetto Zigbee è 127 bytes, mentre il numero limite di bytes attualmente trasferibili tra chiavetta e host è 64 bytes, è necessario accumulare i dati provenienti dalla chiavetta CC2531 fino alla completa ricezione di un frame, che avviene quando il flusso di byte letti è minore di 64 bytes. In Figura 3.6 è possibile osservare la rappresentazione in codice di quanto appena descritto.

```
framedata = []

while True:
    pdata = None
    try:
        pdata = self.dev.read(CC2531.USB_EP4_IN, self._maxPacketSize, timeout=timeout)
    except usb.core.USBError as e:
        if e.errno != 110: # Operation timed out
            print("Error args: {}".format(e.args))
            raise e
        else:
            return None

    # Accumulate in 'framedata' until we have an entire frame
    for byteval in pdata:
        framedata.append(struct.pack("B", byteval))
```

Figura 3.6: Codice driver di lettura dati da CC2531

Il firmware ZBOSS utilizza nativamente un preambolo che viene inserito in testa ai pacchetti che vengono inviati alla macchina host. Pertanto, è stato utilizzato per verificare la corretta sincronia dei frame ricevuti.

Ulteriori controlli si assicurano che la lunghezza del frame sia compatibile con quella specificata nel campo LEN del frame stesso.

I pacchetti che superano i controlli sono incapsulati in una struttura dati che permette la loro manipolazione dal framework KillerBee.

3.5.2 Trasmissione

In questo Paragrafo viene descritta l'implementazione del driver KillerBee per permettere al framework di utilizzare il CC2531 per inviare pacchetti all'interno della rete Zigbee.

La logica del framework KillerBee prevede l'implementazione di una funzione *inject* all'interno del driver, avente il compito di comunicare con il dispositivo hardware in modo da permettere la trasmissione di un pacchetto specificato come parametro.

Per garantire la corretta interazione tra host e dispositivo è, inizialmente, necessario segnalare alla chiavetta CC2531 che un pacchetto sta per essere inviato. In questo modo il CC2531 entra in fase di trasmissione e inizia ad

accumulare i byte trasferiti dall'host fino alla ricezione di un frame completo, come descritto nel Paragrafo 3.4.2. Successivamente, il pacchetto può essere inviato, preoccupandosi di distribuire l'operazione in due fasi diverse nel caso in cui la sua lunghezza sia superiore a 64 bytes, dato il massimo numero di bytes trasferibili per ogni scrittura sul dispositivo. La procedura qui spiegata è stata implementata come in Figura 3.7.

```
# set minimum delay to 0.0007, send-read change status time
# needed for zbstumbler
if delay < 0.0007:
    delay = 0.0007

if len(packet) < 1:
    raise Exception('Empty packet')
if len(packet) > 125: # 127 - 2 to accommodate FCS
    raise Exception('Packet too long')

if channel != None:
    self.set_channel(channel, page)
if page:
    raise Exception('SubGHz not supported')

for pnum in range(0, count):
    # tell the usb that a packet is going to arrive
    self.dev.write(CC2531.USB_EP4_OUT, _to_array(CC2531.USB_SEND_PACKET), 100)

    # upper limit per transfer is 64 bytes
    if len(packet) <= 64:
        self.dev.write(CC2531.USB_EP4_OUT, packet, 100)
        # TODO: manage len(packet)==64
    else:
        self.dev.write(CC2531.USB_EP4_OUT, packet[:64], 100)
        self.dev.write(CC2531.US4B_EP4_OUT, packet[64:], 100)
    time.sleep(delay)
```

Figura 3.7: Codice driver di scrittura dati a CC2531

Come osservabile in Figura 3.7, è possibile indicare il canale Zigbee su cui trasmettere i pacchetti oppure utilizzare quello precedentemente impostato se non viene specificato.

3.6 Test delle nuove funzionalità

In questa sezione saranno mostrati alcuni degli attacchi forniti dal framework KillerBee, servendosi della chiavetta CC2531 su cui è stato installato il nuovo firmware open source descritto nel Paragrafo [3.4](#). La compatibilità del nuovo firmware con KillerBee è resa possibile attraverso l'implementazione dei driver, come mostrato nel Paragrafo [3.5](#).

Il funzionamento dei comandi servirà a testare il funzionamento delle estensioni apportate a KillerBee e al CC2531.

3.6.1 Componenti hardware utilizzati

Il dispositivo al centro dell'analisi è il CC2531 con firmware open source ZBOSS esteso come descritto nel Paragrafo [3.4](#).

Inoltre, per avere una comprensione più profonda e dettagliata dei pacchetti che vengono trasmessi sulla rete Zigbee è stata utilizzata una seconda chiavetta CC2531, su cui è installato il firmware originale della Texas Instruments [\[22\]](#), che permette di usare il dispositivo come packet sniffer. Questo secondo CC2531 permetterà di mostrare il traffico Zigbee generato dalla prima chiavetta.

Per la simulazione della rete Zigbee si è utilizzata una topologia semplice composta da un telecomando IKEA [\[33\]](#) che funge da Coordinatore e da una lampada IKEA [\[33\]](#) che funge da Zigbee End Device.

Per il test di alcune funzionalità si è aggiunta una nuova rete Zigbee composta da un interruttore dimmer Philips e una lampada Philips [\[34\]](#).

3.6.2 Test attacchi KillerBee

Per una descrizione più ampia degli attacchi KillerBee utilizzati, riferirsi al Paragrafo [2.1.2](#).

zbid

Utilizzando il comando *zbid* è possibile enumerare le interfacce dei dispositivi compatibili con KillerBee. La chiavetta CC2531 con il nuovo firmware è stata collegata alla macchina host ed è visibile dal framework KillerBee come testimoniato dall'esito del comando *zbid* in Figura 3.8. Ciò conferma la compatibilità tra KillerBee e la nuova versione del firmware ZBOSS implementata nel Paragrafo 3.4.

```
kevin@kevin-dock:~$ sudo zbid
[sudo] password for kevin:
      Dev Product String      Serial Number
      1:7 CC2531 USB CDC      001
kevin@kevin-dock:~$
```

Figura 3.8: Test comando *zbid*

zbstumbler

Per testare il comando *zbstumbler* si sono create entrambe le reti Zigbee descritte nel Paragrafo 3.6.1. Le lampade sono state poste con il rispettivo interruttore o telecomando in parti diverse dell'abitazione per ricreare un ambiente reale. In Figura 3.9 è possibile osservare l'esito del comando. In particolare, sono state trovate le due reti Zigbee precedentemente create, come ci si aspettava. Il parametro *-i* specificato è superfluo nel caso sia presente un solo dispositivo compatibile con KillerBee. È invece necessario per identificare il destinatario del comando quando gli hardware sono molteplici.

Poichè il funzionamento del comando *zbstumbler* implica la corretta trasmissione di pacchetti *beacon requests*, come mostato nel Paragrafo 2.1.2, è dimostrato che la chiavetta CC2531 è ora in grado di trasmettere pacchetti all'interno della rete Zigbee.

```
kevin@kevin-dock:~$ sudo zbstumbler -i 1:18
zbstumbler: Transmitting and receiving on interface 'CC2531 USB CDC'
An entire packet arrived
New Network: PANID 0x5954 Source 0x0002
                Ext PANID: 09:70:da:c5:e6:be:8b:72      Stack Profile: ZigBee Enterprise
                Stack Version: ZigBee 2006/2007
                Channel: 20
An entire packet arrived
New Network: PANID 0x722D Source 0x0002
                Ext PANID: eb:f8:c0:4c:6b:e1:1e:33      Stack Profile: ZigBee Enterprise
                Stack Version: ZigBee 2006/2007
                Channel: 25
^C
16 packets transmitted, 2 responses.
kevin@kevin-dock:~$
```

Figura 3.9: Test comando zbstumbler

Inizialmente il comando *zbstumbler* non forniva i risultati previsti. Il problema è stato individuato nell'istantaneo passaggio da trasmissione a ricezione pacchetti. È stato necessario inserire un tempo di ritardo tra l'invio di dati e la ricezione di essi per permettere alla chiavetta CC2531 di inviare la *beacon request* correttamente e mettersi in ascolto del *beacon* in risposta. Il tempo di ritardo deve essere il più piccolo possibile per evitare rallentamenti ma, allo stesso momento, sufficientemente grande per garantire un corretto passaggio da trasmissione a ricezione. Il valore è stato sperimentalmente individuato pari a $700\mu s$ per la macchina host su cui si stava lavorando [\[1\]](#). Dal momento che modifiche al codice di *zbstumbler* di KillerBee non sono possibili perchè influenzerebbero il comportamento di tutti i dispositivi hardware utilizzati, l'implementazione del ritardo è stata inserita nel metodo *inject* del driver KillerBee per il CC2531 come mostrato in Figura [3.7](#).

zbwiresnark

Per il test di questo comando si sono utilizzate entrambe le chiavette CC2531 predisposte come descritto nel Paragrafo [3.6.1](#). Entrambe le chiavette sono state messe in ascolto utilizzando il comando *zbwiresnark* sullo stesso canale Zigbee. In Figura [3.10](#) è possibile osservare parte del traffico

¹Linux Ubuntu 18.04 LTS, Intel® Core™ i7-7500U CPU @ 2.70GHz × 4 , 8 GB RAM

Zigbee catturato dai due dispositivi hardware. Nella finestra posizionata a sinistra è mostrato il traffico intercettato dalla chiavetta CC2531 con firmware originale della Texas Instruments mentre, a destra, quello catturato dalla chiavetta CC2531 avente il nuovo firmware ZBOSS esteso. Il traffico registrato è lo stesso per entrambi i dispositivi. Ciò permette di dimostrare il corretto funzionamento dell nuovo firmware, descritto nel Paragrafo 3.4, come packet sniffer.

No.	Time	Source	Info
17	33.467625	0x0002	Command, Dst: Broadcast, Src: 0x0002
18	33.633826	0x0001	Data Request
19	33.634784	0x0001	Ack
20	34.662895	0x0001	Data Request
21	34.663486	0x0001	Ack
22	44.109469	0x0001	Data, Dst: Broadcast, Src: 0x0001
23	44.110199	0x0001	Ack
24	44.121572	0x0001	Data, Dst: Broadcast, Src: 0x0001
25	45.136937	0x0001	Ack
26	45.137798	0x0001	Data Request
27	46.160942	0x0001	Ack
28	46.167797	0x0001	Data Request
29	47.193343	0x0001	Ack
30	47.194195	0x0002	Command, Dst: Broadcast, Src: 0x0002
31	48.027195	0x0002	Command, Dst: Broadcast, Src: 0x0002
32	48.029138	0x0002	Command, Dst: Broadcast, Src: 0x0002
33	79.660153	0x0002	Command, Dst: Broadcast, Src: 0x0002
34	94.054489	0x0002	Command, Dst: Broadcast, Src: 0x0002
35	109.764876	0x0002	Command, Dst: Broadcast, Src: 0x0002
36	126.787248	0x0002	Command, Dst: Broadcast, Src: 0x0002
37	138.389632	0x0001	Data, Dst: Broadcast, Src: 0x0001
38	138.389486	0x0001	Ack
39	138.403232	0x0001	Data, Dst: Broadcast, Src: 0x0001
40	139.416192	0x0001	Data Request
41	139.417101	0x0001	Ack
42	140.442294	0x0001	Data Request
43	140.443372	0x0001	Ack
44	141.469626	0x0001	Data Request
45	141.470532	0x0001	Ack
46	144.262994	0x0002	Command, Dst: Broadcast, Src: 0x0002

Frame 32: 47 bytes on wire (376 bits), 47 bytes captured (376 bits) on interface 0
 IEEE 802.15.4 Data, Dst: Broadcast, Src: 0x0002
 ZigBee Network Layer Command, Dst: Broadcast, Src: 0x0002

Frame 21: 47 bytes on wire (376 bits), 47 bytes captured (376 bits) on interface 0
 IEEE 802.15.4 Data, Dst: Broadcast, Src: 0x0002
 ZigBee Network Layer Command, Dst: Broadcast, Src: 0x0002

Figura 3.10: Test comando zbwire shark

Il traffico catturato è cifrato. È possibile utilizzare la procedura descritta nel Paragrafo 1.2.2 per provare a trovare la Network Key, necessaria per la decifrazione dei pacchetti.

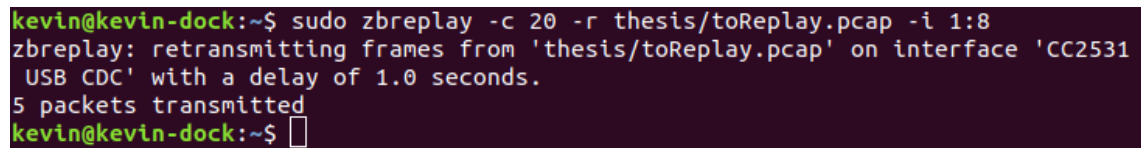
zbreplay

Il comando *zbreplay* implementa un attacco replay su reti Zigbee. Si è già dimostrato con il comando *zbstumbler* nel Paragrafo 3.6.2 che la chiavetta CC2531 con il nuovo firmware è in grado di trasmettere pacchetti correttamente. Lo scopo è ora cercare di portare a termine un attacco replay per spegnere una lampadina Zigbee. La vittima designata è la lampada IKEA

predisposta come descritto nel Paragrafo [3.6.1](#).

In una prima fase, è stato utilizzato il comando *zbireshark* per catturare i pacchetti cifrati relativi allo spegnimento della lampadina. L'intenzione è di ritrasmettere questi pacchetti, provocando uno spegnimento della lampadina IKEA, senza conoscere la Network Key.

In Figura [3.11](#) è possibile osservare il comando utilizzato, a cui è stato passato come parametro il file pcap contenente il traffico precedentemente catturato. Il comando è funzionante ma non ha portato al risultato sperato.



```
kevin@kevin-dock:~$ sudo zbreplay -c 20 -r thesis/toReplay.pcap -i 1:8
zbreplay: retransmitting frames from 'thesis/toReplay.pcap' on interface 'CC2531
USB CDC' with a delay of 1.0 seconds.
5 packets transmitted
kevin@kevin-dock:~$
```

Figura 3.11: Test comando zbreplay

Per comprendere più dettagliatamente le ragioni del mancato successo, si è ripetuto l'attacco mentre la seconda chiavetta CC2531 osservava il traffico effettivamente trasmesso. Come è possibile notare dalla Figura [3.12](#), i pacchetti sono correttamente mandati dal CC2531 e ricevono gli ACK dalla lampadina IKEA. Quest'ultima, tuttavia, non si spegne.

Il motivo di tale comportamento è stato attribuito alle contromisure implementate nel protocollo Zigbee per evitare questo tipo di attacco. In particolare è stato introdotto un campo *Frame Counter* per tracciare il numero del pacchetto. Un dispositivo Zigbee ricevente, controlla che il *Frame Counter* del pacchetto arrivato sia superiore all'ultimo valore visto e lo scarta silenziosamente se così non fosse [\[6\]](#).

No.	Time	Source	Info	Destination	Protocol	Length
1	0.000000	0x0001	Data, Dst: Broadcast, Src: 0x0001	Broadcast	ZigBee	49
2	0.000327		Ack		IEEE 8...	5
3	0.001344		Ack		IEEE 8...	5
4	1.002155	0x0001	Data, Dst: Broadcast, Src: 0x0001	Broadcast	ZigBee	49
5	2.002491	0x0001	Data Request	0x0002	IEEE 8...	12
6	2.003072		Ack		IEEE 8...	5
7	2.003557		Ack		IEEE 8...	5
8	3.003879	0x0001	Data Request	0x0002	IEEE 8...	12
9	3.004398		Ack		IEEE 8...	5
10	3.004806		Ack		IEEE 8...	5
11	4.006335	0x0001	Data Request	0x0002	IEEE 8...	12
12	4.006988		Ack		IEEE 8...	5
13	4.007466		Ack		IEEE 8...	5

Figura 3.12: Traffico tra CC2531 e lampada IKEA durante attacco replay

Un ultimo tentativo è stato effettuato modificando il *Frame Counter* in modo che fosse superiore a quello correntemente utilizzato dal dispositivo Zigbee ricevente (la lampada IKEA), nel tentativo di aggirare le contromisure. Per raggiungere tale scopo si è utilizzato l'editor di testo Vim in modalità esadecimale e sono stati aggiornati i valori esadecimali all'interno del pacchetto corrispondenti al *Frame Counter*.

Tuttavia, neanche questo tentativo ha avuto successo data la presenza del campo *MIC*, *message integrity check*, all'interno del *Security Header* che si occupa di controllare l'integrità del pacchetto [6].

Pertanto, in assenza della Network Key, il comando `zbreplay` non offre alcuna possibilità di successo per i pacchetti Zigbee che dispongono del *Security Header*.

Conclusioni e sviluppi futuri

In accordo con quanto prefissato inizialmente, è stata implementata una soluzione per garantire compatibilità tra il framework KilleBee e il dispositivo hardware CC2531, estendendo le funzionalità di intercettazione del traffico e permettendo di utilizzare la chiavetta CC2531 per trasmettere pacchetti all'interno della rete Zigbee.

Per raggiungere tale obiettivo si è utilizzato il firmware open source fornito da ZBOSS a cui sono state aggiunte le funzionalità di trasmissione dati ricercate. Il nuovo firmware, che inizialmente non disponeva di alcun supporto da parte del framework KillerBee, è stato adattato per lo scopo e sono stati implementati i driver utilizzati dal framework per la compatibilità con il dispositivo hardware.

L'implementazione del nuovo firmware trattata in questo documento apre le porte a nuove prospettive: l'utilizzo di un dispositivo a basso prezzo e costantemente reperibile nel mercato, come il CC2531, per lo studio completo della sicurezza di reti Zigbee, utilizzando il framework professionale KillerBee. Inoltre, il firmware è open source e aperto a chiunque voglia apportare cambiamenti o implementare nuove funzionalità.

Nell'immediato futuro saranno apportati miglioramenti al firmware, non effettuati per mancanza di tempo. In particolare si cercherà di eliminare il tempo di attesa necessario per il passaggio da trasmissione a ricezione dati, descritto nel Paragrafo [3.6.2](#). Inoltre, si provvederà a ottimizzare il codice del firmware per eliminare il passaggio di alcuni dati ridondanti al driver di KillerBee e si gestirà la trasmissione di pacchetti di dimensione pari a 64

bytes da driver a firmware, non attualmente supportata.

Il codice del firmware e dei nuovi driver scritti per KillerBee saranno resi pubblici per permettere a chiunque fosse interessato, di poter utilizzare la soluzione implementata.

Un lavoro interessante potrebbe riguardare lo studio in profondità delle vulnerabilità della rete Zigbee, utilizzando le funzionalità offerte da KillerBee e servendosi di uno o più dispositivi CC2531, adesso in grado di trasmettere pacchetti.

Bibliografia

- [1] Sito web di Zigbee Alliance . <https://zigbeealliance.org/>. Acceduto il 23/06/2020.
- [2] Sito web di Zigbee Alliance. Zigbee specification. <https://zigbeealliance.org/wp-content/uploads/2019/12/docs-05-3474-21-0csg-zigbee-specification.pdf>. Acceduto il 27/06/2020.
- [3] NXP Laboratories. ZigBee 3.0 Stack User Guide. <https://www.nxp.com/docs/en/user-guide/JN-UG-3113.pdf>. Acceduto il 27/06/2020.
- [4] Sito web di Electrical Technology. ZigBee Technology. <https://www.electricaltechnology.org/2017/09/zigbee-technology-wireless-networking-system.html>. Acceduto il 27/06/2020.
- [5] Joshua Wright e Johnny Cache. *Hacking exposed wireless: wireless security secrets and solutions, third edition*. 2015.
- [6] Articolo di Silicon Labs sulla sicurezza del protocollo Zigbee. <https://www.silabs.com/documents/public/application-notes/an1233-zigbee-security.pdf>. Acceduto il 26/06/2020.
- [7] Tobias Zillner. ZIGBEE EXPLOITED The good, the bad and the ugly. <https://www.blackhat.com/docs/us-15/materials/us-15-Zillner-ZigBee-Exploited-The-Good-The-Bad-And-The-Ugly.pdf>. Acceduto il 28/06/2020.

-
- [8] Twitter. Profilo di MayaZigBee. <https://twitter.com/MayaZigBee>.
Acceduto il 28/06/2020.
 - [9] Sito web di PeeVeeOne. ZLL Commissioning trust centre link key. <https://peeveeone.com/?p=166>. Acceduto il 28/06/2020.
 - [10] Pagina Github di KillerBee. <https://github.com/riverloopsec/killerbee>. Acceduto il 20/06/2020.
 - [11] Pagina Github di Z3sec. <https://github.com/IoTsec/Z3sec>. Acceduto il 20/06/2020.
 - [12] P. Morgner, S. Matthejat, Z. Benenson, C. Müller, F. Armknecht: *Insecure to the Touch: Attacking ZigBee 3.0 via Touchlink Commissioning*. Boston 2017.
 - [13] Pagina Github di SecBee. <https://github.com/Cognosec/SecBee>.
Acceduto il 20/06/2020.
 - [14] Pagina Github di Zigdiggity. <https://github.com/BishopFox/zigdiggity>. Acceduto il 21/06/2020.
 - [15] Atmel RZ RAVEN USB stick. <https://www.element14.com/community/docs/DOC-67532/1/avr-rz-usb-stick-module>. Acceduto il 21/06/2020.
 - [16] Datasheet di TelosB mote. http://www.memsic.com/userfiles/files/Datasheets/WSN/telosb_datasheet.pdf. Acceduto il 21/06/2020.
 - [17] Pagina Github del prodotto Apimote di RiverLoop. <https://github.com/riverloopsec/apimote>. Acceduto il 22/06/2020.
 - [18] Open Sniffer di Sewio. <https://www.sewio.net/open-sniffer/>.
Acceduto il 23/06/2020.

-
- [19] CC2531 Dongle USB della Texas Instruments. <https://www.ti.com/tool/CC2531EMK>. Acceduto il 23/06/2020.
- [20] Sito web di Amazon. <https://www.amazon.it/>. Acceduto il 30/06/2020.
- [21] Sito web di AliExpress. <https://www.aliexpress.com/>. Acceduto il 30/06/2020.
- [22] CC2531 Packet sniffer della Texas Instruments. <https://www.ti.com/tool/PACKET-SNIFFER>. Acceduto il 23/06/2020.
- [23] Sito web di Zigbee2mqtt. <https://www.zigbee2mqtt.io/>. Acceduto il 23/06/2020.
- [24] CC2531 Z-STACK firmware della Texas Instruments. <https://www.ti.com/tool/Z-STACK>. Acceduto il 23/06/2020.
- [25] CC2531 CC Debugger della Texas Instruments. <https://www.ti.com/tool/CC-DEBUGGER>. Acceduto il 23/06/2020.
- [26] Guida installazione firmware con strumenti alternativi. https://www.zigbee2mqtt.io/getting_started/flashing_the_cc2531.html. Acceduto il 23/06/2020.
- [27] Sito web di Raspberry Pi. <https://www.raspberrypi.org/>. Acceduto il 23/06/2020.
- [28] Sito web di Arduino. <https://www.arduino.cc/>. Acceduto il 23/06/2020.
- [29] Sito web di ZBOSS . <https://dsr-zboss.com/#!/>. Acceduto il 23/06/2020.
- [30] Sito web di ZBOSS per sviluppatori. <https://zboss.dsr-wireless.com/>. Acceduto il 23/06/2020.

- [31] Datasheet del chip CC2531 . <https://www.ti.com/lit/ug/swru191f/swru191f.pdf>. Acceduto il 24/06/2020.
- [32] Sito web di IAR Embedded Workbench per CPU 8051 . <https://www.iar.com/iar-embedded-workbench/#!?architecture=8051>. Acceduto il 24/06/2020.
- [33] Catalogo lampada e telecomando IKEA. <https://www.ikea.com/it/it/p/tradfri-kit-con-telecomando-spettro-bianco-20406570/>. Acceduto il 25/06/2020.
- [34] Catalogo interruttore dimmer e lampada Philips. <https://www2.meethue.com/it-it/p/hue-white-wireless-dimming-kit-e27/8718696785331>. Acceduto il 25/06/2020.

Ringraziamenti

Vorrei ringraziare il Prof. Ing. Marco Prandini per il supporto che mi ha fornito, non solo in questa fase finale del percorso ma anche durante la mia esperienza Erasmus.

Un grazie ai miei correlatori di tesi. In particolare, volevo ringraziare l'Ing. Davide Berardi per la disponibilità, i consigli e le direzioni che ha saputo indicarmi.

Ringrazio la mia famiglia per tutto ciò che mi ha insegnato, non sarò mai grato abbastanza.

Grazie ai miei amici, che mi hanno sempre perdonato le sere in cui li ho dovuti abbandonare per portare avanti i miei studi.