ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

Dipartimento di Informatica – Scienza e Ingegneria

TESI DI LAUREA

in

Sicurezza Dell'Informazione M

Progettazione e realizzazione in ambiente Internet of Things di un sistema di sicurezza con approccio security by design e metodologie agile

CANDIDATO
Giovanni Ciandrini

RELATORE
Chiar.ma Prof.ssa Rebecca Montanari

CORRELATORE
Dott. Ciro Donato Caiazzo

Anno Accademico 2016/17
Sessione II

A mio nonno Claudio

"Viaggiatore, non esiste sentiero. Il sentiero si costruisce mentre cammini"

(Antonio Machado: 1875-1939)

Indice

Introduzione

L'informatica e le tecnologie stanno cambiando radicalmente. I trend tecnologici più importanti del momento stanno per decollare (Sistemi di Intelligenza Artificiale con strumenti di machine/deep learning, App Intelligenti, Realtà Virtuale e Aumentata, Elaborazione del Linguaggio e conversazioni intelligenti, Internet of Things, Blockchain e architetture decentralizzate), trasformando tanti aspetti della vita quotidiana. Da qualche anno si sta affermando il trend Internet of Things, che fa riferimento alla possibilità di andare a connettere in rete qualsiasi oggetto sia dotato di caratteristiche minime per essere indirizzabile, andando potenzialmente a modificare tanti aspetti nella vita di tutti i giorni.

Infatti, la possibilità di costruire reti *smart*, dominate da oggetti e sensori in grado di scambiare dati con l'esterno, apre scenari innovativi, come applicazioni software in grado di gestire: gli aspetti di una casa, quali sistema antifurto e comfort; gli aspetti di una macchina, quali manutenzione olio e freni; gli aspetti di un'industria, con la manutenzione dei macchinari; gli aspetti di una città, con le informazioni legate al traffico e alla criminalità, ma anche al ribilanciamento del carico elettrico; gli aspetti di un ospedale, con il monitoring delle condizioni di un paziente e dei macchinari associati ai diversi reparti…

Tante aziende vogliono investire in questo trend che è considerato molto propizio e che Gartner stabilisce essere di riferimento per i prossimi anni, con la previsione che nel 2020 ci siano circa 25 miliardi di dispositivi connessi in rete: per poter investire su di una soluzione innovativa di questo tipo occorre affrontare però delle tematiche e delle sfide tutt'oggi dominanti nel settore.

Se le tematiche legate al mercato possono essere affrontate con un'analisi accurata e le sfide legate alla tecnologia e alla scalabilità dei prodotti IoT possono essere affrontate con un accurato progetto tecnologico, la tematica che pone un grosso limite all'ascesa piena di questo trend è senza dubbio l'inesistenza di standard e soluzioni per la sicurezza dei dati e dei dispositivi.

La sicurezza delle cose (Safety) e la sicurezza delle reti e dei dati (Security) per la prima volta nella storia dell'informatica diventano un'unica cosa con IoT, con l'amplificazione della gravità delle conseguenze di una violazione: un attacco di manomissione ad una telecamera che gestisce un sistema di antifurto, un attacco ad un pacemaker in un ospedale o un attacco ad un sensore sensibile in una smart city può portare conseguenze molto gravi che arrivano a minare l'incolumità fisica di persone. Nonostante questo, in IoT non è previsto alcun meccanismo preventivo di difesa.

Oltre alle possibili gravi conseguenze di un attacco, in IoT vengono continuamente immessi sul mercato sensori fisici completamente vulnerabili che utilizzano protocolli di comunicazione che non sono stati affatto messi in sicurezza, permettendo ad avversari di sfruttare nuovi access point quando questi sensori vengono collegati in rete, arrivando a effettuare attacchi molto importanti che compromettono interi sistemi e intere città (basti pensare al blackout in ucraina con il virus Black Energy, a fine 2015, o al virus Mirai del 2016).

Con l'evidente difficoltà teorica di trovare un'unica soluzione standard di sicurezza dei dati e dei dispositivi per un problema così profondo e dinamico, in un mondo che fa dell'eterogeneità un suo

fattore chiave, occorre stabilire dal design di un sistema quali sono gli obiettivi da garantire per una soluzione software IoT-sicura: visibilità piena della soluzione IoT, sviluppo dinamico e veloce, e ovviamente criteri di sicurezza da cablare nel comportamento del sistema stesso.

L'obiettivo di questo lavoro di Tesi è stato definire un flusso di lavoro all'interno del reparto di Ricerca e Sviluppo per andare a valutare i benefici e i limiti di una particolare tecnica di inserimento di requisiti di sicurezza, che fa riferimento al mondo Agile, chiamata *abuser stories,* applicandola pienamente ad un prototipo reale IoT, cercando di valutare se attraverso questa tecnica sia possibile inserire i requisiti di sicurezza di un sistema software by design e quali vantaggi e limiti portano alla definizione di un completo sistema di sicurezza per un certo requisito.

La metodologia di sviluppo Agile garantisce come valori la visibilità, la trasparenza e l'adattabilità necessaria per un sistema software che si deve integrare con una rete preesistente: con Agile è possibile quindi parlare concretamente di security by design, ovvero andare a definire i requisiti di sicurezza sin dal principio, attraverso il cablaggio di questi requisiti nel comportamento del software stesso. Per quanto riguarda la tematica sicurezza, in questo progetto si esplora una metodologia Agile particolare, chiamata *abuser stories*, che descrive una modalità precisa per andare ad inserire il fattore sicurezza in un sistema software costruito attraverso l'ecosistema Agile.

Nel primo capitolo viene introdotto il trend IoT, i possibili domini di applicazione e il problema della sicurezza come fattore abilitante di questo trend. Nel secondo capitolo verrà presentata l'architettura di riferimento IoT general-purpose e i vari scenari di attacco a questa architettura, iniziando a proporre alcune tecniche per la messa in sicurezza di questi livelli; verranno inoltre presentati i protocolli di comunicazione utilizzati in IoT, in particolar modo concentrandosi sullo stato dell'arte del Bluetooth Low Energy, e si illustrerà infine la metodologia Agile e in che maniera affrontare un'integrazione della sicurezza nell'ecosistema di lavoro. Nel terzo capitolo verrà applicata la metodologia in questione su un prototipo IoT aziendale come caso di studio specifico: verrà analizzata, progettata, realizzata e mantenuta una soluzione di sicurezza, motivando a fondo ogni scelta di progettazione, integrandola infine attraverso degli spunti con analitiche di Machine Learning e di gestione di Big Data; verrano valutate poi le conclusioni e i risultati di questa metodologia nella sua applicazione pratica. Nell'appendice viene illustrato più concretamente il dettaglio tecnologico dei vari punti affrontati nel lungo percorso di questo progetto di ricerca.

Capitolo 1: Architetture e scenari per Internet of Things

1.1 Internet Of Things e contesto

1.1.1 Definizione, percorso e scenari applicativi

Con Internet Of Things (IoT) si fa riferimento a un sistema di dispositivi computazionali (*things*) interconnessi tra di loro, identificabili e abilitati a trasferire dati su una rete, senza una necessaria interazione umana [ROU, 2016]. Il concetto di Thing può essere in pratica potenzialmente riferito a qualsiasi oggetto in grado di essere identificato, tramite il possesso di un indirizzo IP, e capace di trasferire dati: si parla di interazione *machine-to-machine* (M2M), ossia di interazioni tramite oggetti che non hanno mai avuto capacità computazionale (impianti elettrici, frigoriferi, orologi, lavatrici, pacemakers, pompe di insulina...). Tante esigenze differenti, come ad esempio la possibilità di gestire un impianto di antifurto in casa, gestire la manutenzione di macchinari industriali, i freni di una macchina, la ridistribuzione elettrica in una grande città… potrebbero creare tante soluzioni di prodotto, consolidando IoT come un trend dove coesistono differenti scenari applicativi e innumerevoli spunti che un'azienda può cogliere per definire e realizzare prodotti e soluzioni in questo trend di mercato, che risulta essere propizio nel breve termine. Gli scenari applicativi di questa innovazione sono molteplici [KAS, 2016], tra cui:

- *Smart Home*
  È lo scenario IoT più accattivante: il concetto si riferisce all'interazione potenziale di tanti oggetti domestici, controllati via pc o oppure via smartphone, nell'ottica ad esempio di regolare l'accensione dell'aria condizionata prima di arrivare a casa, oppure per lo spegnimento delle luci dopo aver lasciato la propria abitazione, o per gestire ad esempio un sistema di antifurto: il costo è alto, ma questi prodotti promettono in teoria di risparmiare nel lungo periodo tempo, energia e investimenti.

- *Wearable*
  Tante multinazionali hanno investito in questo ambito IoT. I dispositivi *wearable* sono oggetti indossabili dotati di sensori e software che collezionano dati e informazioni riguardo gli utenti; dati che poi vengono elaborati in una seconda fase per estrarre informazioni anche per la profilazione di questi utenti. Parliamo di dispositivi che riguardano soprattutto

l'ambito fitness e salute, accomunati dall'utilizzo di tecnologie efficienti e a basso consumo di energia.

- *Connected Car*
  Una macchina connessa (chiamata anche *smart car*) è un veicolo che è in grado di ottimizzare le sue operazioni attraverso sensori on board e connessione a Internet, come ad esempio manutenzione e garanzia di comfort per i passeggeri. Molte aziende automobilistiche e start up stanno lavorando su questo ambito, anche in relazione all'imminente avvento nel breve-medio termine delle macchine ibride come mercato di riferimento [GHI, 2015].

- *Industrial Internet of Things*
  IIoT è un nuovo concetto nel settore industriale, termine declinato anche in *Industry 4.0*, dove si assume che le macchine robotiche e industriali dotate di sensori per la raccolta dei dati possano essere molto accurate e veloci nella comunicazione ed elaborazione di dati: dati che possono aiutare le aziende sulla gestione dei problemi, soprattutto riguardo malfunzionamenti e manutenzioni, in real-time. In questo ambito troviamo anche applicazioni volte a tracking di oggetti, scambio informazioni real-time sui canali di retail e gestione di consegne automatizzate, in ordine di potenziare l'efficienza della catena di distribuzione.

- *Smart Cities*
  Un'altra applicazione molto importante in IoT è il concetto di Smart City. Scenari applicativi in questo ambito comprendono, attraverso l'utilizzo di dati raccolti da diversi sensori, mansioni automatizzate come sorveglianza urbana, trasporto, gestione dell'energia, distribuzione dell'acqua, sicurezza urbana. Si cercano quindi di risolvere i problemi più importanti di persone che vivono in città, come la congestione del traffico e la gestione del carico elettrico (altro concetto, *Smart Grid*), il tutto attraverso sensori e l'uso di applicazioni web.

Le possibilità dell'IoT, quindi, portano già nel 2017 questa tecnologia a un grande successo che potrebbe essere pienamente raggiunto solamente tramite la risoluzione delle sfide attualmente aperte nel settore, accomunate in molti casi dalla ricerca e da sforzi di standardizzazione per un mondo che fa dell'eterogeneità il suo fattore chiave. È all'interno di questo trend che si giocheranno senza alcun dubbio nel breve termine la maggior parte di nuovi investimenti, soluzioni e proposte: considerando tutti questi aspetti, è naturale pensare che la sfida per la supremazia sul mercato sia ancora aperta, e soprattutto che è molto importante svolgere progetti di ricerca all'interno di questo caldo ambito.

IoT ha avuto un percorso storico che parte da lontano[BRE, 2016]: da *ArpaNet* (1969), prima rete di computer ad uso militare in guerra fredda, all'invenzione dello stack *TCP/IP* (1974) e del successivo *DNS* (1989), passando per l'invenzione del *World Wide Web* (1989) e la nascita di *Google* (1998), la prima volta che fu usata l'espressione "Internet delle cose" era il 1999, termine

coniato dal ricercatore Kevin Ashton, ed è proprio in questi anni che l'insieme di concetti e di spunti per le soluzioni IoT vennero messe insieme per prepararsi all'innovazione che avrebbe stravolto i decenni successivi del mondo informatico. Nel periodo 2000-2010 IoT inizia a diventare un termine di riferimento nel mondo scientifico, e più precisamente è dal 2008 che è possibile decretare la nascita del termine non solo come concetto teorico, ma anche come reale e fisica interconnessione tra internet e gli oggetti, e di conseguenza tra le persone, con un numero di dispositivi collegati ad Internet di 12,5 miliardi, record mai toccato prima. Sarà poi nel 2011 con la nascita del protocollo *Ipv6* che i servizi offerti sulla rete verranno ampliati e la gestione degli indirizzi IP semplificata. L'IoT fino al 2017 ha prodotto reddito pari a 655 miliardi di dollari, ed entro il 2025, con l'ipotesi di 20 miliardi di dispositivi collegati nel 2020[SIL, 2015], si può facilmente comprendere che l'IoT giocherà un ruolo economico fondamentale, con una stima dell'11% di reddito sull'economia mondiale. Attualmente l'IoT racchiude una costellazione di tecnologie e soluzioni differenti, tutte a un certo punto di maturità di innovazione, come evidenziato nell'hype cycle di Gartner per le tecnologie IoT del 2016 (si veda Figura 1).

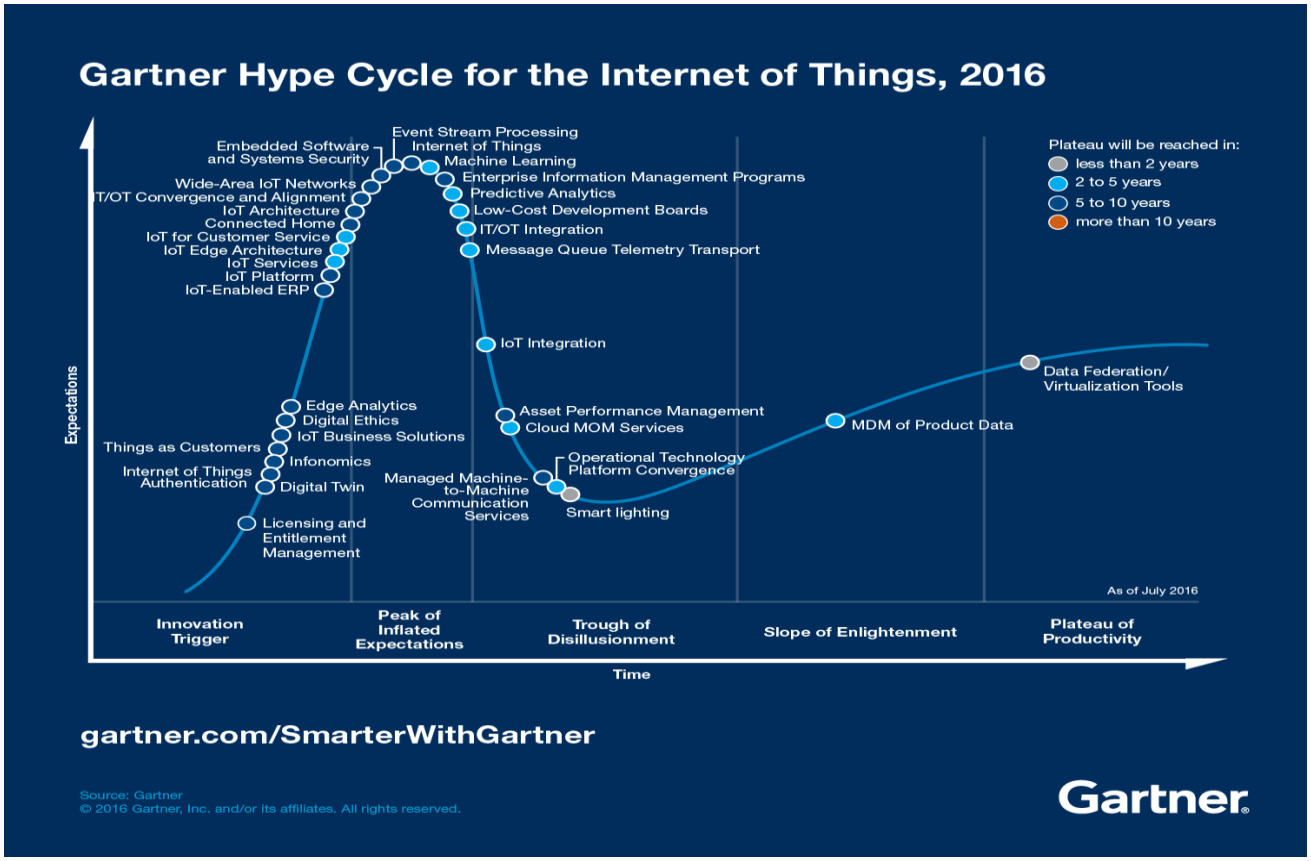

Figura 1: Hype Cycle Gartner per IoT, 2016

Il mondo IoT è costellato da diverse sfumature tecnologiche, che raggiungeranno il plateau non oltre i prossimi 5 anni. Quasi la maggior parte delle tecnologie infatti sono nel picco delle aspettative (Machine Learning, IoT Architecture, Sicurezza dei sistemi e sistemi Embedded), a conferma di come la tecnologia IoT, anche se ha già raggiunto un larghissimo bacino di utenti e di investimenti, verrà delineata proprio nei prossimi 3,4 anni (2017-2020), in relazione a come saranno affrontate e risolte sfide importanti e fondamentali per la stabilità della gran parte di soluzioni IoT da parte di aziende, ricercatori,università. Il trend si è affermato unendo diverse singole tappe

storiche che furono separate tra di loro sfruttando il loro collegamento, e questo può far presagire che le sfide che questa tecnologia pone agli sviluppatori e ai ricercatori di tutto il mondo, in relazione all'eterogeneità della sua natura e dei suoi problemi, debbano essere affrontate unendo più mondi e concetti già affrontati e maturi, in un punto di vista complessivo magari completamente nuovo e re-ingegnerizzato.

1.1.2 Certezze e sfide aperte nell'Internet of Things

Tutti gli aspetti principali della tecnologia partono ovviamente dalla definizione di un'**architettura** per questo mondo, che già rappresenta un vero e proprio dilemma proprio a causa della sua natura frammentaria. Si parte identificando i due concetti chiave nell'Internet of Things [FRE, 2015]: Things *(*dispositivi) e Server-Side Architecture *(*supporto per i dispositvi). Parliamo di un vero supporto server-side (e non solo di Internet) perché molto spesso abbiamo uno o più altri elementi architetturali, che fanno da ponte tra i dispositivi sensoristici (che quasi sempre sono entità a basso consumo che non sono in grado di esporsi autonomamente sul web) e il vero e proprio Server che rappresenta la connessione a internet. Questo elemento aggiuntivo è spesso impersonato da uno o più Gateway a bassa potenza, che a seconda dello scenario e della funzionalità di una certa soluzione IoT, può svolgere funzioni di aggregazione, processare eventi, oppure fare semplicemente bridging tra diversi dispositivi e Internet, come si nota nella figura 2.

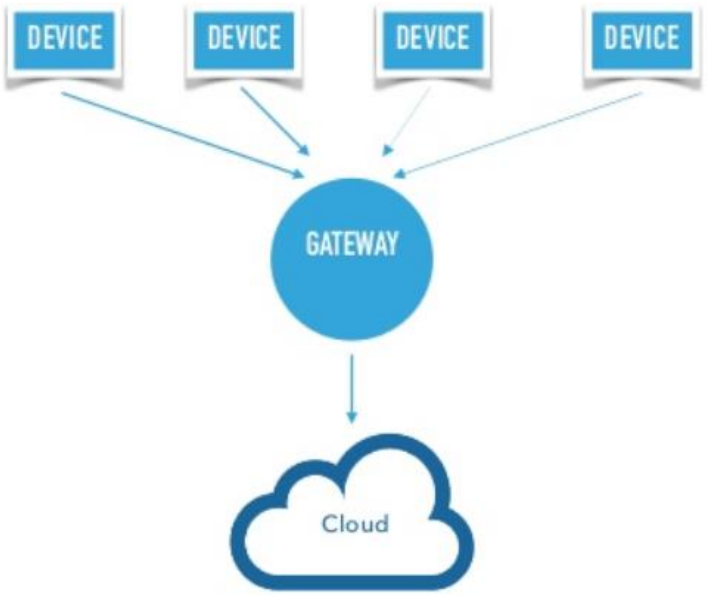

Figura 2: Architettura Internet of Things General-Purpose

Se tantissime soluzioni IoT potrebbero essere rappresentate da una configurazione a tre livelli come descritto, attualmente in realtà non esiste una vera e propria architettura orizzontale di riferimento: ogni vendor cerca di proporre la propria piattaforma IoT cloud (Amazon, Microsoft) che è ottimizzata per una specifica funzionalità, in termine di mantenere una buona "scalabilità" verso il numero sempre maggiore di utenti/dispositivi. Abbiamo eterogeneità e punti aperti su tutto [SAL,

2015]: dai protocolli di comunicazione possibili che è possibile usare tra i dispositivi (Ethernet o Wi-Fi, Bluetooth o Bluetooth Smart, NFC, ZigBee, UART), ai dispositivi fisici che possono essere a loro volta eterogenei (da SOC controllers di 8 bit come Arduino, arrivando a vere e proprie piattaforme computazionali a 32 o 64 bit, come Raspberry PI), ai diversi protocolli che si possono utilizzare per comunicare sul cloud (MQTT, COAP, WebSocket, COAP). Un'architettura di riferimento comprenderebbe uno sforzo di standardizzazione elevato, e forse impossibile, ma il percorso stesso per provare a trovare una soluzione omogenea potrebbe risolvere o comunque approfondire alcuni punti aperti nel mondo IoT. Dopo la descrizione di un primo scoglio architetturale, si identificano tre possibili categorie dove occorre affrontare delle sfide, come per ogni nuovo trend tecnologico [BAN, 2017].

La prima sfida si gioca nel campo del **business,** dove è fondamentale avere un modello solido per identificare le motivazioni per investire in un certo prodotto oppure no, e capire quali applicazioni possono monetizzare maggiormente rispetto ad altre e in quali tempi, suddividendo i prodotti in *Consumer IOT (*connected car, wearables), *Commercial IOT* (IOT Healtcare, Smart City) e *Industrial IOT*, con business plan mirati verso il settore in cui un'azienda vuole entrare. La seconda si gioca nel campo **society**, dove occorre immedesimarsi nella prospettiva del consumatore affrontando diversi risvolti: un cambio continuo di requisiti e di domanda sul mercato, nuovi usi e bisogni del cliente, differente confidenza soggettiva verso un prodotto piuttosto di un altro, e soprattutto mancanza di comprensione e di best practices da parte del consumatore medio verso un trend tecnologico apparentemente sconosciuto e innovativo, in relazione all'uso "scorretto" dei prodotti IoT in particolar modo riguardo ambito privacy e sicurezza. La **privacy,** infatti, è una delle sfide più importanti all'interno dell'ambito di sicurezza IOT, ed è fondamentale capire che non riguarda esclusivamente sistemi e contromisure tecnologiche, ma che gran parte della sicurezza e della protezione di dati sensibili si gioca sul comportamento degli utenti: tanti sistemi IoT hanno l'obiettivo di entrare in maniera pervasiva negli ambienti privati degli utenti, con il fine di raccogliere una discreta quantità di dati per elaborare sempre meglio informazioni importanti, utenti che nella maggior parte dei casi sono totalmente inconsapevoli di come questi dispositivi siano integrati con la sfera privata. Il concetto di privacy è legato fortemente alla modalità con cui le aziende acquisiscono, elaborano, espongono e trattano dati, che molte volte hanno una natura di forte sensibilità (dati geolocalizzati, dati relativi a salute, di movimento); per questo motivo una regolarizzazione normativa è necessaria, e con l'entrata in vigore della normativa europea per il trattamento dei dati personali (ECTS, stabilita dal 2016 ma che entrerà in vigore nel Maggio 2018 [GAR, 2016]), tanti di questi temi vengono affrontati e finalmente regolarizzati, stabilendo piena trasparenza e piena consapevolezza di come aziende e multinazionali dovranno trattare dati relativi agli utenti, andando a standardizzare e a decidere uniformemente le modalità di gestione di dati che fino ad oggi rimanevano opache e dipendenti dai singoli vendor. Infine si ha la categoria dove convivono le sfide più importanti, che sono lasciate agli "addetti ai lavori", sviluppatori e Ingegneri: le sfide tecnologiche per creare sistemi e soluzioni IOT.

Tanti aspetti interdisciplinari, sfide, ricerche e investimenti convivono in questa categoria, e i temi affrontati sono molteplici e tutti in uno stato completamente aperto:

- **Connectivity & Scalability:**

Connettere un enorme numero di dispositivi è una delle sfide più importanti per l'IoT. Oggi ci si affida molto spesso ad architetture centralizzate, nodi server/client, ma già in tante soluzioni (e nel futuro, con miliardi di dispositivi da gestire) il primato sarà ottenuto da chi riuscirà ad offrire un'architettura efficiente e totalmente decentralizzata: parliamo ad esempio di *fog computing* [CAS, 2017], *peer-to-peer*, e in generale soluzioni che hanno a loro volta delle sfide da affrontare, una su tutte la sicurezza;

- **Compatibility and standard:**
  Occorre avere la piena consapevolezza che non esiste né un'unica architettura né ovviamente un unico standard per le soluzioni IoT: come già espresso, l'IoT cresce in differenti direzioni, e non essendoci protocolli standard per l'interazione M2M avremo diversità sia a livello di firmware dei dispositivi, sia a livello di protocolli di rete. Tante sfide sono anche dovute al fatto che i dati dei dispositivi rientrano quasi completamente nella sfera **Big Data,** ovvero dati descritti in ordine di 5V (Velocity, Veracity, Volume, Variety, Value), totalmente differenti da dati relazionali e quindi sono necessarie nuovi modelli e architetture per la loro elaborazione/memorizzazione (Data Warehouse, Hadoop, Spark);

- **Intelligent Analysis & Actions:**
  Estrarre aspetti comportamentali e informazioni dai dati per delle analisi, ovvero una vera e propria analitica intelligente di dati che provengono da dispositivi IOT, è un aspetto che determina la vera natura dell'ambito applicativo IoT ma allo stesso tempo costellato di sfide, ad esempio nella gestione real-time di un largo volume di dati, passando da una "batch analysis" a una "streaming analysis", anche in relazione alla gestione dell'interoperabilità tra i vari dispositivi e, ancora una volta, in ottica di sicurezza dell'informazione e privacy.

Se da una parte quindi è vero che verticalmente abbiamo tante possibilità di sviluppo di soluzioni IOT per i campi più disparati, questa varietà molto spesso non supporta l'interoperabilità fra modelli e dispositivi. Il campo che rappresenta forse la tematica più calda in ambito IoT e in generale tecnologico, con un'importanza pienamente trasversale, è senz'altro quello relativo alla **sicurezza IOT** e **cybersecurity**, forse il tema più abilitante verso una diffusione piena di prodotti IOT in scala mondiale. Il mondo della sicurezza informatica in IoT sta cambiando decisamente, a partire dagli usi nella propria abitazione fino ad arrivare al luogo di lavoro [BEL, 2017]. Si riassumono in figura 3 le sfide tecnologiche in IoT.

Figura 3: Le sfide tecnologiche in Internet Of Things

1.2 IOT e Security

1.2.1 Il problema della Sicurezza in IOT: "The S in IOT stands for Security"

Si può riassumere in questo breve aforisma quasi tutta la tematica sicurezza per quanto riguarda questo trend tecnologico. L' IoT, per quanto a livello applicativo sia una delle innovazioni più importanti della storia della tecnologia, non solo manca completamente di un supporto standard alla sicurezza (come gran parte delle soluzioni tecnologiche moderne), ma il suo avvento ha reso ancora più vulnerabile e debole questo aspetto. Le numerose problematiche potrebbero essere riassunte in alcuni punti principali, che descrivono le cause e le gravi conseguenze di uno scarso supporto alla sicurezza all'interno di IoT [EAS, 2017]:

- **Crescita Esponenziale e nuove vulnerabilità**:
  Il processo di aumento esponenziale di dispositivi connessi [SIL, 2015], se da una parte rappresenta proprio l'unicità del trend, esso ne contribuisce all'aumento di vulnerabilità come una vera e propria arma a doppio taglio. In particolar modo occorre considerare che si passa dal dover proteggere singoli computer (o dispositivi mobili), al doverne proteggere *n* differenti (automobili, termostati, webcam, frigoriferi, pacemakers, sensori, orologi, condizionatori), ognuno con un proprio livello di sicurezza interno (molte volte pari a zero). Occorre considerare inoltre che andando a collegare tanti dispositivi in rete aumentano a dismisura gli access point attraverso i quali un avversario potrebbe sfruttare un qualsiasi tipo di attacco (classico e non) con l'intento di entrare all'interno di queste reti per un certo obiettivo malevolo;

- **Safety and Security:**
  La mancanza di un livello di sicurezza interno ai dispositivi è uno degli aspetti chiave in questo campo: quasi tutti i dispositivi vengono implementati con software minimo per soddisfare i requisiti applicativi per cui vengono usati, che molto spesso non solo non

prevede alcuna misura di sicurezza (software e dispositivi rilasciati velocemente), ma oltretutto questo software non può essere aggiornato. Dato che nella storia della sicurezza informatica le tappe fondamentali della lotta tra difensori e attaccanti sono state quasi sempre segnate dal rilascio di uno o più aggiornamenti, in ordine di proteggere server aziendali associati ad esempio a un eventuale rischio economico, oggi nell'IoT non solo con dispositivi rigidi è pressoché impossibile realizzare aggiornamenti o patch, ma andando a connettere in rete sensori domestici, linee elettriche, pacemakers e altri dispositivi "sensibili" la cui *safety* (sicurezza) viene gestita direttamente tramite internet, occorre chiedersi: cosa potrebbe fare un attaccante una volta preso possesso di un dispositivo o una volta entrato in una rete personale di una *smart home*? Quali informazioni, quali dati può vedere, manipolare e operazioni svolgere? Quanto siamo effettivamente al sicuro?
Non è un caso che questi dispositivi siano spesso sprovvisti di un supporto alla sicurezza: essi in molti casi vengono lanciati sul mercato nel minor tempo possibile per sfruttarne un vantaggio competitivo; tanti altri invece hanno una bassissima capacità computazionale, e quindi diventa infattibile progettare anche un leggero protocollo o supporto di sicurezza, aprendo un'ulteriore sfida sulla gestione di modalità di sicurezza a basso consumo di energia.
Safety (sicurezza "fisica" delle cose) e *Security* (sicurezza informatica, dei sistemi), che sono sempre stati concetti separati fino ad oggi, nell'IoT convergono nello stesso piano, e nel caso di attacco o di violazione al sistema le conseguenze nell'IoT potrebbero non solo essere rischi economici ma allargarsi in qualcosa di molto più grave.

- **Eterogeneità e vulnerabilità di codice:**
  Non solo tanti dispositivi utilizzati oggi nell'IoT non prevedono un vero e proprio livello di sicurezza interno, ma un ulteriore problema è che anche se disponiamo di dispositivi abbastanza sicuri internamente, l'IoT è un mondo così variegato, eterogeneo e in costante crescita che occorre mettere in sicurezza le comunicazioni che avvengono tra diversi dispositivi all'interno di una stessa applicazione , considerando soprattutto che quando comunichiamo verso Internet potremo utilizzare n protocolli, m configurazioni e k dispositivi diversi (con n,m,k valori molto alti), ognuno con le sue caratteristiche e vulnerabilità di sicurezza. Una stessa applicazione IoT consolidata potrebbe aver bisogno, ad esempio in ottica di una migliore scalabilità o di nuove funzionalità, di una fase di refactoring in termini di protocolli, dispositivi e configurazioni utilizzate, e tutto ciò non può che opporsi a un requisito di sicurezza applicativo standard che per poter andare a garantire un certo tipo di proprietà (confidenzialità, integrità, autenticazione) ha bisogno, come minimo, di una configurazione abbastanza fissa o al limite dinamica in maniera prevedibile. Anche se i dispositivi, l'architettura e le interazioni fossero pienamente prevedibili e al sicuro, è fondamentale che anche il codice stesso che viene eseguito sui dispositivi sia sicuro, e nell'IoT non è banale nemmeno questo passaggio, considerando l'eterogeneità tra le piattaforme e soprattutto i differenti livelli di programmabilità dei dispositivi: anche se avessimo l'architettura IoT più sicura al mondo, una sola istruzione non sicura farebbe crollare tutto. Per quanto riguarda le soluzioni di sicurezza sul cloud, è abitudine affidarsi alle soluzioni che i vendor propongono: anche in questo caso occorre valutare e delineare

quali requisiti di sicurezza garantire, a quale profondità, e soprattutto come si comportano le aziende una volta raccolti dati personali e spesso sensibili.

- **Protezione dati dalle aziende e comportamento degli utenti:**
  Come già espresso occorre considerare che nell'IoT una minaccia su tutte è proprio rappresentata dalle aziende che gestiscono e propongono soluzioni IoT sul mercato.
  Che cosa fa un'azienda una volta raccolti i dati di trasferimenti monetari attraverso uno smartphone? Come si comporta un'azienda con dei dati relativi alla salute di un certo paziente, o relativi a credenziali per conti online? La privacy è un argomento fondamentale da affrontare in ambito sicurezza, sia intesa come sicurezza che i nostri dati siano al sicuro, sia intesa come sicurezza che l'azienda tratti i nostri dati in maniera riservata.
  Gran parte delle questioni in sospeso saranno effettivamente regolamentate nel *GPDR (ECTS)* [GAR, 2016] che entrerà in vigore nel maggio 2018, e quindi si va verso una soluzione omogenea e trasparente per il consumatore. Occorre comunque tenere a mente che la migliore protezione che l'utente ha per quanto riguarda dati sensibili e privati è sempre quella di leggere attentamente ogni tipo di contratto prima di accettare delle condizioni.
  Comportamento degli utenti che è decisivo in generale su tutte le situazioni di sicurezza, e nell'IoT più che mai: la prima linea difensiva in una smarthome o in una qualsiasi rete IoT gestita da un consumatore si gioca proprio nell'applicare semplici best-practices, che spesso rappresentano le prime vulnerabilità ricercate da un attaccante. Cambiare le password di default di un dispositivo, usare dispositivi aggiornati o comunque configurabili in termini di sicurezza, non andare a collegare tanti dispositivi a un solo end-point (dove magari controlliamo conti online), sono un esempio di un primo esempio di buone abitudini.
  È naturale che un consumatore medio non può poi prendersi sulle sue spalle tutta la sicurezza del proprio prodotto IoT, ed a un certo punto dovrà affidarsi a qualcun altro, gettandosi nel suo mare: la differenza è che se vengono applicate alcune semplici best-practices, il tuffo lo fa da un'altezza molto più bassa.

Riassumendo, parlare di un'unica soluzione a un problema così profondo, dinamico ed eterogeneo è azzardato: proprio in questi anni molti paesi internazionali, nonostante un atteggiamento da sempre refrattario e in generale di poca cultura sull'argomento sicurezza informatica, hanno iniziato a delineare e investire su veri e propri piani d'azione relativi alla sicurezza informatica[RIG, 2016], per quanto riguarda ad esempio la protezione di informazioni e dei dati dei cittadini, di istituzioni, e molto altro. La tematica sicurezza sta diventando, grazie anche a tecnologie come l'IoT, un punto di discussione fondamentale e da affrontare a tutti i livelli di ogni ambito sociale, anche se ancora oggi la poca cultura e conoscenza su questo settore e il suo grande impatto sull'industria fa sì che gli investimenti siano decisamente molto bassi e la situazione di gran lunga sottovalutata dalle imprese e dalle nazioni, specialmente in Italia: si veda figura 4.

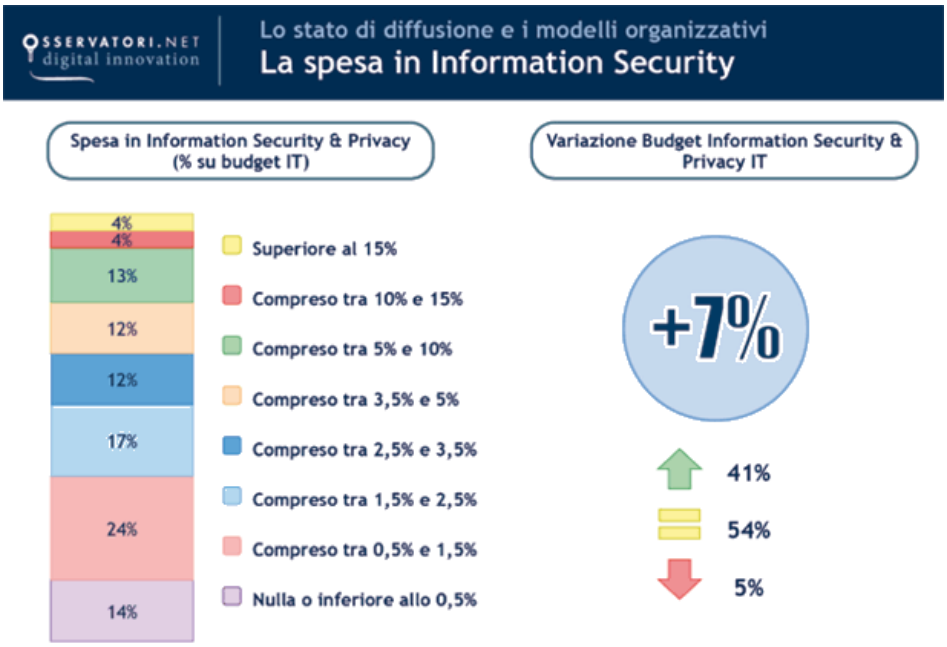

Figura 4: Studio del politecnico di Milano del 2016 su dati relativi a investimenti in Italia sulla Information Security

Trovare una soluzione standard per la sicurezza nell'IoT significherebbe avere a che fare con una configurazione architetturale standard nell'IoT: configurazione standard che potrebbe non essere mai trovata, a causa della sua natura eterogenea e della sua natura applicativa adattativa. Anche se questa configurazione architetturale condivisa non dovesse essere mai trovata, è invece necessario trovare una soluzione per il problema sicurezza: una soluzione per la sicurezza deve giocarsi all'interno di un vero e proprio piano d'azione aziendale, che riguarda non solo la condivisione di informazioni all'interno di più reparti di un'azienda, ma che verte anche sul decidere quali livelli di tradeoff scegliere per i diversi aspetti di un prodotto (performance / sicurezza , standardizzazione / eterogeneità , security / manutenzione), e soprattutto si gioca sul mettere insieme concetti e metodologie tecnologiche che magari non sono nate con l'intento di mettere in sicurezza delle soluzioni IoT, ma che oggi, e soprattutto nel futuro, potrebbero dare una grossa mano verso la stabilizzazione non di un'unica soluzione per un mondo eterogeneo, ma di best-practices eventualmente automatizzate e il più possibile indipendenti da protocolli, configurazioni e architetture (che possono cambiare) per garantire, in diversi scenari architetturali, requisiti di sicurezza comuni per una soluzione IoT.

1.2.2 Soluzioni attuali e direzione per il futuro

Scenari inediti che accompagnano il trend IoT non possono che richiedere soluzioni innovative per quanto riguarda la sicurezza: il problema è che alcuni scenari sembrano delinearsi solamente oggi,

rimanendo ancora delle soluzioni di "nicchia" e pioneristiche, confermando però quanto sia aperto attualmente il settore e nello stesso tempo fondamentale da approfondire. Su quali aspetti e proprietà occorre concentrare gli sforzi in chiave sicurezza? È consigliabile riusare concetti, tecniche e configurazioni di sicurezza che sono servite in passato? Occorre delineare strutturalmente un percorso e un flusso di ragionamento per affrontare questa problematica, magari basandosi su concetti e proprietà chiave con la quale costruire una proposta di soluzione. La realtà è che un prodotto IoT in termini di rete non può essere considerato dal punto di vista della sicurezza come indipendente e separato dal resto del mondo, poiché esso interagisce con la rete preesistente, con tutti gli end-point, cloud e sistemi IT sia fisici che virtuali: strategie individuali di sicurezza IoT finiscono solo per aggiungere elementi, e ridurre la capacità di visione. Con una tipologia di approccio organica si potrebbe permettere di ottenere **visibilità** sull'intero ecosistema di reti: ecco qual è il primo aspetto chiave da considerare.

Scendendo ad un dettaglio più tecnico[LON, 2017], potrebbe essere un buon inizio partire dalle tecnologie a disposizione: per quanto riguarda i dispositivi, soprattutto in ambito wearable[BOW, 2017], in alcune soluzioni moderne vengono usati per lo più dispositivi rigidi che assicurano una sorta di integrità attraverso firmware sicuro ma difficilmente modificabile, in grado poi eventualmente a livello di privacy di essere versatile, fornendo soluzioni dirette o tramite interfaccia con cui gli utenti possono gestire in maniera trasparente la propria privacy. Un'altra proprietà di sicurezza fondamentale per questi dispositivi è la protezione delle identità e degli accessi: a questo livello, occorre pensare anche ad archivi di identità che consentono di memorizzare le informazioni del dispositivo dell'utente, includendo già in una prima fase di analisi modelli di database, in particolare riferendosi a database *NoSQL* che, in questo scenario, potrebbero avere le caratteristiche necessarie al fine di memorizzare ed elaborare tipologie di dati provenienti da questi dispositivi[ASA, 2014]. Per quanto riguarda l'autenticazione, ci sono anche soluzioni specifiche utilizzate anche in passato, come lo IAM (Identity and Access Management [MOR et al., 2016]), e inoltre altro punto fondamentale riguarda garantire la confidenzialità, con soluzioni di crittografia IoT utilizzando algoritmi di crittografia standard e utilizzando la PKI (Infrastruttura a chiave pubblica) per fornire certificati digitali X.509. Oltre le proprietà classiche di sicurezza, in questo nuovo trend è fondamentale anche avere strumenti di analisi di sicurezza IoT (raccolta, aggregazione, monitoring e normalizzazione dei dati per fornire report), e nel migliore dei casi, modalità con la quale rilevare minacce IoT (ad esempio, identificare attacchi wireless basati sull'instabilità della rete IoT). E' naturale pensare che questi approcci alla sicurezza, che in gran parte riprendono considerazioni del passato sulla sicurezza, se da una parte sono essenziali da considerare come punto di partenza una volta identificate le proprietà da garantire, dall'altra parte si scontrano inesorabilmente con i **limiti intrinseci** dei dispositivi connessi: i nodi terminali hanno strutturalmente pochissime risorse sia in termini di potenza di calcolo che di energia disponibile, questo significa che sarebbe quasi impraticabile implementare su questi dispositivi meccanismi o algoritmi di crittografia classici (che appesantirebbero oltremodo la computazione). In relazione alla proprietà di fault tolerance, inoltre, potrebbe essere una buona idea segmentare la rete: se da una parte come già espresso occorre avere piena visibilità di come la rete IoT interagisce con tutti gli attori di un'architettura IT, dall'altra occorre progettare il sistema in maniera tale da riuscire ad isolare le sezioni a rischio e limitare i danni, per mantenerli ad un livello gestibile. Una volta identificati requisiti di sicurezza in fase di progettazione della propria soluzione IoT, occorre senz'altro adeguare i sistemi di sicurezza tradizionali a una realtà dove i dispositivi sono rigidi (con

un firmware quindi difficile da cambiare), dotati di risorse strutturalmente assai limitate e pienamente integrati nella rete: siamo ancora molto distanti da una soluzione di questo tipo, ma la direzione dovrà essere delineata tenendo conto di queste considerazioni.

In questo ambito così variegato, occorre contestualizzare precisamente quali potrebbero essere gli scenari critici di un prodotto IoT a differenza di altri, e non c'è altro modo che tenere conto di tutti questi fattori già a livello prototipale e di progettazione del software, anche per poter implementare soluzioni che siano il più facilmente isolabili possibili, nell'ottica di un eventuale attacco informatico: diventa fondamentale come non mai in questo contesto il concetto di **security by design**, ovvero l'inserimento della sicurezza come vero e proprio requisito funzionale nel processo di vita del software, a partire dalla sua nascita. Processo di vita del software che oltre a riguardare gli sviluppatori del prodotto in questione dovrà quindi coinvolgere analisti di sicurezza e anche referenti aziendali, andando a definire passo dopo passo quali tradeoff, quali considerazioni e quali aspetti di sicurezza garantire, valutare e verificare, analizzando accuratamente ogni fattore che potrebbe infierire su una decisione.

Tantissime ricerche (universitarie e non) sono state presentate negli ultimi anni in ordine di stabilire una soluzione il più generale possibile: se nel passato la sicurezza è stata sempre studiata in relazione al livello di astrazione dello stack ISO/OSI a cui facevamo riferimento (sicurezza livello di rete, sicurezza livello di trasporto, applicativo), in letteratura oggi viene proposta un'architettura cross-layered [SIN et al., 2015] che superi questo limite, andando a ribadire che l'innovazione della tecnologia IoT è rappresentata dalla proposta di un vero e proprio stack architetturale su più livelli, e che quindi serva una soluzione in grado di gestire la comunicazione e la sicurezza su tutti i livelli OSI, in un completo scenario di integrazione di sistemi mobili nella rete. Elementi e studi di cross-layering che ritroviamo anche nelle numerose ricerche che trattano la gestione della problematica della sicurezza in dispositivi a basso consumo di energia, che nella maggior parte dei casi possiedono pochissime risorse computazionali: in altre pubblicazioni [GRA et al., 2015] viene ribadita l'esigenza di trovare soluzioni di sicurezza per questi dispositivi in ordine di abilitare la loro totale integrazione sulla rete, e viene in particolar modo evidenziata la tematica della gestione delle chiavi, che rappresenta il punto fondamentale di un qualsiasi protocollo di autenticazione o confidenzialità, lasciando aperti tanti spunti e possibilità di ricerca. Per quanto è fondamentale dunque oggi integrare sicurezza sin dall'inizio del progetto di vita di un software IoT, il software deve continuare ad essere comunque incrementale e soprattutto entrare velocemente nel mercato, in uno scenario di **fast development,** per ottenere i benefici maggiori della tecnologia: il punto focale sta nell'identificare quindi un equilibrio tra la velocità con cui questi prodotti possono essere presentati sul mercato, e un supporto in fase di progettazione (magari automatico) che garantisca che il software IoT che è costruito sia sicuro in relazione ai requisiti di sicurezza desiderati, e anche in ottica di nuove vulnerabilità che potrebbero esserci nella soluzione IoT.

Partendo dalle proprietà identificate in relazione alla sicurezza in IoT (visibilità, limiti computazionali, security by design, fast development), occorre analizzare quale potrebbe essere la strada giusta per trovare un supporto di sviluppo che garantisca queste proprietà. In questo contesto lo sforzo sarà quello di ricercare una metodologia di progettazione del software che garantisca uno sviluppo incrementale, iterativo, ma che comprenda anche proprietà di sicurezza sin dalla progettazione, e magari aperto alla manutenibilità e all'estendibilità. La metodologia di lavoro

Agile[BEC et al., 2001] (con particolare riferimento a Scrum e tecniche di XP come TDD[ROU, 2017]), garantisce tutto ciò che riguarda la progettazione di un software che rispecchi le proprietà elencate, a meno dell'aspetto di security: sarà dunque lo sforzo del progetto di ricerca integrare la sicurezza nella progettazione Agile di una soluzione IoT, che magari dia adito a un supporto non solo facilmente usabile e interoperabile, ma soprattutto estendibile e mantenibile, verso tecniche anche innovative di gestione della sicurezza come il **Machine Learning.** Infatti, nonostante tanti attuali limiti intrinseci e ancora forte instabilità, il Machine Learning potrebbe nel breve futuro essere la chiave per andare ad affrontare il problema sicurezza nell'IoT [SHA, 2017]. In particolare, la sua ascesa potrebbe essere legata al fatto che l'incremento esponenziale di nuovi sensori integrati e integrabili sulla rete potrà abilitare un grande flusso di dati, in base ai quali algoritmi e tecnologie di apprendimento lavorano, fornendo potenzialmente supporti in grado di determinare automaticamente se qualche tipologia di azione malevole è in corso (ad esempio per quanto riguarda la *network intrusion detection*); analisi di dati che sarebbe ingestibile da elaborare a livello umano in un tempo accettabile.

In questo scenario totalmente aperto ed innovativo, il progetto di ricerca avrà l'ambizione di andare concretamente a ricercare, progettare, definire ed infine di implementare su di un caso di studio aziendale un supporto software in grado di fornire una soluzione innovativa e pioneristica per queste problematiche, seppur a livello prototipale.

Capitolo 2: Metodologie di progettazione e tecnologie di riferimento IoT

2.1: Architettura Internet of Things e supporto alla sicurezza

2.1.1: Stack architetturale IoT

Negli ultimi due anni (2015-2017) sono stati proposti diversi stack architetturali per cercare di dare una visione globale per un sistema software IoT: trovare una configurazione di riferimento sarà, come già espresso, una strada abilitante per la futura definizione di standard tecnologici e soluzioni vincenti nel mercato applicativo di queste tecnologie.

In particolare, in letteratura [KHA et al., 2012] si considera generalmente uno stack IoT a tre livelli, come quello mostrato in figura 5: Application, Network e perception. Con *application* intendiamo il livello applicativo, ovvero dove vengono effettivamente implementati i software che offrono un servizio (per esempio in un ambiente Web); con *network* intendiamo il livello di comunicazione della rete, dove gli oggetti effettivamente vengono connessi per un servizio IoT, e che dunque diventa di fondamentale importanza considerare; infine, con *perception* intendiamo il livello fisico degli oggetti connessi alla rete (gli oggetti veri e propri).

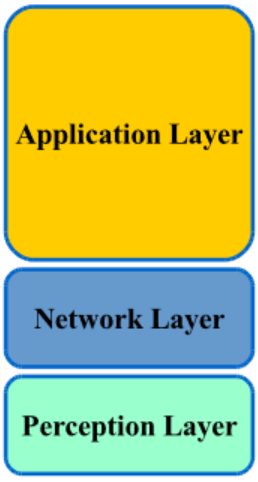

Figura 5: Architettura multi-livello Internet of Things

L'architettura può sempre maggiormente complicarsi: potrebbe esserci un ulteriore livello di business che va a gestire un'astrazione ulteriore verso l'utente finale, ad esempio per integrare dati in un'analitica decisionale (in ottica Business Intelligence, Business Analysis e Big Data); oppure, possono inserirsi livelli di astrazione intermedi. Generalmente, però, si fa riferimento a questa architettura verticale, e dunque ai possibili attacchi che possono presentarsi in questa specifica configurazione.

2.1.2: Possibili attacchi e relativi supporti alla sicurezza

Per l'architettura generale IoT a tre livelli descritta sopra vengono definiti alcuni possibili scenari di attacco ad ogni livello e i relativi supporti alla sicurezza, rimanendo su un'analisi generale che non dipende dall'utilizzo applicativo successivo di questa architettura, ma che mette piena importanza sul partire da scenari malevoli e di attacco nella definizione di uno scenario di sicurezza condiviso. Verranno messi in evidenza durante il discorso gli attacchi che caratterizzano e che è molto importante considerare nel nuovo trend tecnologico. Come primo strumento viene descritto OWASP (Open web application security project) [OWA, 2001], organizzazione e progetto open-source mondiale no-profit nato nel 2001 con l'obiettivo di migliorare la sicurezza informatica degli applicativi, che definisce sotto forma di elenco quali sono i rischi e gli scenari di attacco più comuni all'interno di un applicazione web, andando a raccogliere ed esplicitare gli errori più comuni, gli exploit più utilizzati da attaccanti per prendere ad esempio il controllo del programma, dei dati e del pieno funzionamento di un sistema. Per verificare queste vulnerabilità e scenari di supporto per specifiche tipologie di attacco, Owasp propone anche dei test per verificarli. Negli ultimi anni, con il consolidamento di IoT, Owasp ha definito anche le vulnerabilità più comuni di un'architettura per questo trend, realizzando uno specifico Owasp IoT project [OWA1] e andando a definire le vulnerabilità più comuni all'interno dell'architettura IoT [OWA, 2014]. In particolare, in riferimento all'architettura descritta nel capitolo precedente, si nota come è possibile facilmente associare diversi attacchi ai diversi livelli architetturali di IoT (application, network e perception layer), che in molti casi sfruttano vulnerabilità condivise tra i diversi livelli.

Uno degli esempi concreti di attacco in IoT: tra i primi dieci scenari più comuni di insicurezza IoT si ha ad esempio lo scenario di un'interfaccia web insicura, come nel caso in cui chiunque abbia accesso ad uno specifico login e le eventuali credenziali siano deboli; questa vulnerabilità ha un impatto molto grave sui sistemi IoT e attacchi mirati, come un'interruzione del servizio (D.o.S.) o il furto di identità (e di manomissione del dispositivo). Questo scenario potrebbe essere testato simulando casi reali di utilizzo dell'interfaccia stessa, come l'utilizzo della funzionalità "Recupero Password", o l'implementazione di attacchi molto comuni come SQL-Injection oppure Cross-site Scripting (XSS) sul codice. Infine, vengono elencati in Owasp anche i vari punti sul quale agire per limitare l'attacco in questione, come ad esempio l'impostazione di credenziali robuste, un rinforzo del meccanismo di recupero password, assicurarsi di non essere suscettibili a XSS, SQLI o altri attacchi simili (e ad esempio utilizzare anche qualche tool per analizzare la sicurezza del codice stesso, come SonarQube [SON]). Altri esempi di particolari attacchi in IoT sono: insufficiente autenticazione/autorizzazione, servizi di rete insicuri, mancanza di sistemi di crittografia, aspetti legati alla privacy, software o firmware insicuro, povera sicurezza fisica dei componenti. Tutti questi scenari navigano sui tre livelli di riferimento architetturale IoT. Un altro strumento che elenca gli attacchi IoT è STRIDE [STR], modello di classificazione delle minacce sviluppato da Microsoft. STRIDE è l'acronimo per sei tipologie di attacchi comuni in IoT:

- **S**poofing dell'identità dell'utente
- **T**ampering (manomissione)
- **R**ipudio (autenticazione)
- **I**nformation disclosure (ambito privacy)
- **D**enial of Service (D. o. S)

- **E**levation of Privilege

Oltre all'importanza di definire un buon supporto per la questione privacy con la verifica di un'eventuale *Information disclosure attack*, oppure sviluppare codice sicuro per evitare attacchi classici di *D. o. S*, per il mondo IoT uno scenario molto importante da tenere in considerazione è che il trend è dominato da tantissimi dispositivi fisici, e quindi diventa essenziale gestire l'effettiva sicurezza "fisica" di questi dispositivi. Lo scenario **tampering**, che fa riferimento a una manomissione e danneggiamento fisica di un dispositivo, diventa un requisito fondamentale da tenere in considerazione nella costruzione e nella definizione di una soluzione software per diversi ambiti applicativi in IoT. Dall'healthcare a smart home, da Smart City a Industry 4.0, la manomissione di uno o più dispositivi IoT, oltre che danneggiare il funzionamento dell'architettura software, potrebbe avere anche avere conseguenze molto più gravi: la manomissione di un allarme antifurto, di un sensore di un pacemaker, dei freni di una smart car, sono tutte operazioni che potrebbero avere conseguenze ragionevolmente catastrofiche.

La manomissione di un dispositivo fa parte di una famiglia più grande di attacchi al dispositivo fisico [SOU, 2017], che viene definita come manipolazione, ad esempio di dati all'interno di server, di router, di clienti, manipolazioni per forzare crash di sistema e quindi D.o.S., etc… Inoltre, sono da considerare anche gli scenari di furto, sia fisico del dispositivo che dei dati al suo interno, e quindi di frode, nel momento in cui un avversario si autentica utilizzando credenziali rubate. Tutti questi aspetti caratterizzano esclusivamente la tecnologia IoT, e possono essere limitati con alcune best practices: crittando i dati, utilizzando certificazioni e firme digitali per autenticazione, ma soprattutto mettendo in sicurezza il dispositivo stesso e il cloud su cui i dati vanno a esporsi: il problema però, come già descritto, consiste nell'eterogeneità dei diversi dispositivi che si connettono in rete e nella loro bassa capacità computazionale.

Nonostante i diversi punti critici e la diversità dei diversi ambiti applicativi, si ha un punto in comune molto importante che in un certo senso accresce l'ambizione e la speranza di trovare una modalità concreta di gestire uno specifico scenario di sicurezza in IoT, come lo scenario della manomissione di un dispositivo: le tecnologie e i protocolli che riguardano la comunicazione nei sistemi dove sono inseriti questi dispositivi, infatti, sono usati in tantissimi ambiti IoT. *Bluetooth Low Energy, Wifi, ZigBee*, sono tre dei protocolli di livello fisico di comunicazione più usati da questi dispositivi e, come verrà spiegato, l'importanza di conoscere il protocollo specifico di comunicazione ovviamente può permette di ragionare a fondo sulle tematiche sicurezza offerte da un protocollo piuttosto che un altro, andando a mettere l'aspetto sicurezza all'interno dei punti decisionali della scelta applicativa di un protocollo. E non solo: nel momento in cui viene definito un flusso per la verifica di uno scenario di sicurezza (come la verifica di manomissione di un dispositivo), potrebbe essere possibile, dato il largo utilizzo di questi protocolli di comunicazione, cercare una vera e propria modalità di verifica di questo scenario in diversi prodotti IoT e individuare attraverso dei test se la vulnerabilità è presente oppure no, andando quindi ad agire direttamente su un'eventuale contromisura.

I limiti di questo approccio sono essenzialmente due: per prima cosa, non tutti gli scenari di sicurezza possono essere verificati a priori nella costruzione del software, considerando anche che i dispositivi IoT verranno inseriti in un contesto completamente dinamico e connesso, e quindi diventa imprevedibile definire quali possono essere tutti gli scenari di attacco da considerare e da

verificare, tenendo conto poi dell'enorme crescita poi di malware nel 2017 [UNU et al., 2017]. Inoltre, inizia a diventare veramente costoso andare a gestire un flusso di lavoro sulla sicurezza di questo tipo, soprattutto quando viene svolto (come nella maggior parte dei casi attuali) in fase di manutenzione, se non ad attacco avvenuto, quando implementare sicurezza diventa veramente costoso e complicato (la curva dei costi totali, mostrata in figura, concretizza proprio il modello di questo ragionamento).

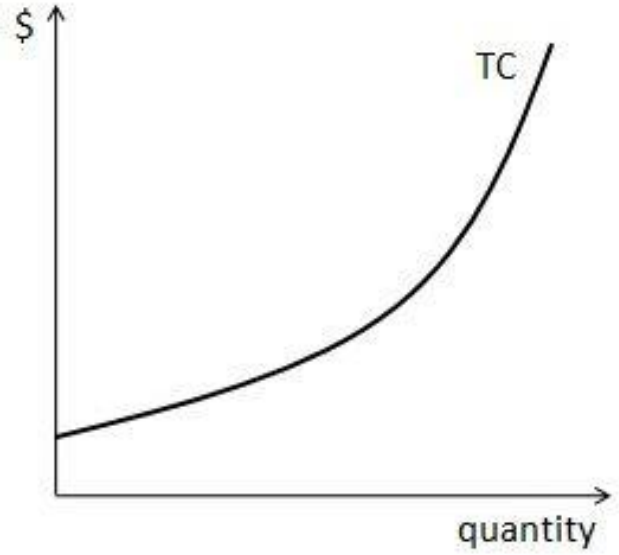

Figura 6: Curva dei costi totali in relazione alla quantità di beni prodotti (o alla durata di un progetto)

2.2: Panoramica protocolli di comunicazione in Internet of Things

L'analisi quindi passa dalla definizione di un primo stack di riferimento IoT allo studio dei protocolli standard che sono stati definiti per questi livelli [ALF et al., 2015], in particolare per quanto riguarda le principali funzionalità di un sistema software: identificazione e comunicazione. Uno dei lavori più importanti di standardizzazione nel mondo IoT che ha coinvolto diverse realtà di ricerca internazionali quali *Internet Engineering Task Force (IETF), World Wibe Web Consodrtium (W3C), Institute of Eletrical and Electronics Engineers (IEEE)* ed *European Telecommunications Standards Institute (ETSI)* è stato quello di provvedere un supporto standard di protocolli di comunicazione per lo sviluppo di applicazioni IoT, come si evidenzia nella Tabella 1.

Application Protocol		DDS	CoAP	AMQP	MQTT	MQTT-SN	XMPP	HTTP REST
Service Discovery		mDNS				DNS-SD		
Infrastructure Protocols	Routing Protocol	RPL						
	Network Layer	6LoWPAN				IPv4/IPv6		
	Link Layer	IEEE 802.15.4						
	Physical/ Device Layer	LTE-A	EPCglobal	IEEE 802.15.4		Z-Wave		

Tabella 1: Protocolli di comunicazione in Internet of Things

Per quanto non è particolarmente interessante andare a valutare effettivamente tutti i possibili protocolli e i vari pro e contro sul loro eventuale uso, è importante evidenziare che per ogni livello di stack IoT abbiamo differenti canali e modalità di comunicazione, con i relativi differenti supporti alla sicurezza. Partendo da un livello fisico di comunicazione dove sono presenti protocolli per lo più efficienti, a basso consumo di energia e che molto spesso rimangono in un'area abbastanza limitata (come ZigBee e Bluetooth Low Energy) con vari problemi riguardo gli aspetti legati alla sicurezza, si passa poi per protocolli di rete (come *IpV6*) ed infine protocolli di livello applicativo (come MQTT, CoAP, AMQP) che in moltissimi casi forniscono alti livelli configurabili di sicurezza, che ad esempio si avvalgono di SSL (Secure Socket Layer), garantendo una comunicazione sicura su reti TCP/IP [HMQ, 2016]. Non è detto che una specifica soluzione IoT debba prevedere tutto lo stack protocollare: molte volte anche solo il collegamento di un piccolo sensore alla rete IT potrebbe bastare per avere una certa funzionalità applicativa, e quasi sempre il collegamento di questo unico sensore potrebbe aprire voragini nella rete e mandare in tilt l'intero ecosistema nella quale è stato inserito. Da qui la conclusione elementare che è praticamente impossibile andare a definire a priori la sicurezza dei protocolli, non conoscendo lo scenario applicativo della soluzione IoT, che di per sé potrebbe presentarsi in tanti modi diversi.

In particolare, tra le più utilizzate tecnologie possibili di comunicazione fisica in IoT si hanno: RFID, 3G, GSM, UMTS, WiFi, Bluetooth Low Energy, ZigBee.

Nella scelta di quale protocollo di comunicazione utilizzare in una specifica soluzione IoT vengono tenuti in considerazione tanti parametri che dipendono dallo scenario applicativo in questione: ad esempio, se uno dei criteri nella scelta dipende dal basso consumo di energia, è possibile restringere questa famiglia di protocolli in un confronto a due tra Bluetooth Low Energy e Zigbee, che sono molto simili ma che differiscono su alcuni punti (si faccia riferimento alla tabella 2).

ZIGBEE	BLUETOOTH LOW ENERGY
Usa la frequenza 2.4 GHz ISM	Usa la frequenza 2.4 GHz ISM
Local Area Network (LAN)	Personal Area Network (PAN)
Throughput: 250 kb/s	Troughput: 270 kb/s
Topologia Mesh	Topologia mesh e a stella
Range: circa 300 metri	Range: circa 80 metri
Potenza di trasmissione: 100 mW	Potenza di trasmissione: 10 mW

Tabella 2: Confronto tra ZigBee e Bluetooth Low Energy

I due protocolli tecnologicamente sono molti simili. Distinguendosi solamente sulla "gittata" del segnale trasmesso, essi possono implementare scenari applicativi IoT differenti. Come casi d'uso specifici alla loro natura, ad esempio: con ZigBee è possibile realizzare applicazioni di home automation, smart lightning, collezione di dati medici, mentre con BLE è possibile gestire invece uno scenario più ristretto del tipo "smart car", come apertura delle porte, connessione con lo smartphone, ma anche più generalmente informazioni sulla salute e sugli allenamenti (pressione arteriosa, temperatura, profilo del battito cardiaco, posizione GPS, velocità di corsa). Si decide di approfondire BLE dato il suo legame con il mondo fit e wearable (ma gli stessi discorsi e il percorso di progettazione valgono anche per una tecnologia più simile a Zigbee).

2.2.1 Bluetooth Low Energy

Bluetooth è uno standard aperto nato nei primi anni 2000 per trasmissioni dati di frequenze radio a corto raggio. La tecnologia Bluetooth è utilizzata prevalentemente nel contesto di reti locali denominate WPANs (Wireless Personal Area Network, IEEE 802.15.x), totalmente differenti per la loro natura da altre topologie di reti, come ad esempio WLANs (Wi-Fi). La tecnologia Bluetooth è stata integrata a partire dai suoi primi anni in tantissimi tipi di business e dispositivi di consumo, tra i quali computer, cellulari, cuffie, stampanti, e negli ultimi anni soprattutto in smartwatches, dispositivi medici, automobili (things). Questo ha permesso agli utenti di creare particolari reti per lo scambio di voce e dati, chiamate reti *ad-hoc o piconet* nel bluetooth, intese come particolare sistema autonomo di terminali connessi mediante collegamenti wireless, ridefinendo durante gli anni il concetto di rete fissa e di routing, presentando una nuova modalità di instradamento dei pacchetti (*multi-hopping*) e introducendo la piena mobilità di tutti gli elementi della rete (dispositivi, router, server). Bluetooth fu subito adottato sin dai primi anni da gran parte delle case produttrici dei vari dispositivi: le varie versioni, oltre che apportare decisivi miglioramenti tecnologici, hanno contribuito ad un delineamento standard anche sulla sicurezza [PAD et al., 2017].

Tra tutte le versioni Bluetooth attualmente la più adottata è la versione 4.0, chiamata anche Bluetooth Smart o più comunemente *Bluetooth Low Energy* (BLE, d'ora in avanti). BLE, proposto

nel 2011, è la versione di Bluetooth che più di tutte ha cercato di ridurre i consumi energetici, con l'obiettivo di aggregare dati provenienti da diversi sensori, come monitor a frequenza cardiaca, termometri, ma anche smart cars, frigoriferi, lavatrici, tramite un'ottimizzazione tecnologica a discapito della velocità di trasmissione. Per la sua natura, la crescita del trend IoT non poteva non passare per l'adozione di un protocollo di livello fisico di questo tipo: ci sono innumerevoli scenari di riferimento in cui la topologia piconet viene adottata all'interno di ambiti IoT [GUT et al. , 2013]. Per quanto riguarda la tematica sicurezza, la tecnologia BLE e i dispositivi su cui è implementata risulta essere pienamente vulnerabile a classiche minacce wireless, come attacchi D.o.S., eavesdropping (ascolto passivo), attacchi MitM (man in the middle), modificazione di messaggi, appropriazione di risorse. Ed ecco che i dispositivi IoT insicuri devono quasi la propria totalità di questa insicurezza al protocollo che utilizzano per esporre dati: attacchi contro dispositivi Bluetooth insicuri possono consentire un accesso non autorizzato a informazioni sensibili (conti in banca, credenziali di un servizio web, stato della salute, stato dell'antifurto) e soprattutto uso non autorizzato dei dispositivi Bluetooth e dei sistemi in cui questi dispositivi vengono integrati (freni di una smart car, pompe di insuline, dispositivi medici di ogni tipo…). Per migliorare la sicurezza dell'implementazione Bluetooth le case produttrici di questi dispositivi dovrebbero implementare delle precise specifiche, dato che Bluetooth fornisce diverse modalità di sicurezza, ma il problema principale risiede nella pressoché impossibilità di andare a configurare questo livello di sicurezza lato sviluppo, livello che viene definito nel sistema operativo e nel firmware da chi produce il dispositivo, solitamente nullo a causa della maggior spesa su altri requisiti (usabilità, scalabilità, time to market), andando a popolare il mercato con dispositivi completamente insicuri e facilmente attaccabili [KUM, 2017]. Inoltre, le diverse modalità che da specifica BLE vengono offerte per la sicurezza sono sostanzialmente quattro livelli di differente granularità, che a loro volta risentono di particolari vulnerabilità.

Dopo aver presentato brevemente le principali caratteristiche tecniche di funzionamento del Bluetooth, ci si appresta a visualizzare quali sono questi supporti alla sicurezza e in quali modi potrebbe essere condotta un'analisi e un processo decisionale sulle modalità di visualizzazione di eventuali vulnerabilità in un dispositivo bluetooth che deve fare parte di un sistema software, e quindi la progettazione di contromisure per mettere in sicurezza alcune di queste vulnerabilità.

Più tecnicamente, BLE consente di costruire a basso costo e con un consumo molto ridotto di energia particolari reti ad-hoc chiamate *piconet*. Come per la tecnologia Bluetooth tradizionale, la *piconet* è una rete composta solitamente da due o più dispositivi Bluetooth vicini tra di loro, che operano sullo stesso canale e utilizzano la stessa sequenza di *frequency hopping*, stabilita in fase di *pairing* (associazione) tra i dispositivi: un esempio classico è il collegamento tra un cellulare e auricolari che utilizzano la tecnologia Bluetooth. Con frequency hopping (FHSS) [FHS] si intende una particolare tecnica che viene utilizzata dal protocollo Bluetooth per le trasmissioni: questa tecnica permette di aumentare la larghezza di banda di un segnale, variando la frequenza di trasmissione a intervalli regolari in maniera pseudocasuale, attraverso un codice prestabilito tra i due nodi della comunicazione. Per ricevere correttamente la trasmissione, il dispositivo bluetooth ricevitore deve quindi conoscere la sequenza esatta dei "salti" di frequenza ed essere completamente sincronizzato con il trasmettitore, al fine di ottenere la sequenza informativa esatta e non dei frammenti sparsi senza alcun significato. FHSS è nato per ridurre gli errori sulle ritrasmissioni dei dati: è importante evidenziare inoltre che esso (assieme a un'altra tecnica,

chiamata “radio link power control”) dà un addizionale ma limitata protezione da ascolto passivo (*eavesdropping*) e accessi malevoli [PAD et al. , 2017]. Diventa infatti complicato per un avversario localizzare e catturare le trasmissioni Bluetooth attraverso l’ascolto di una frequenza fissa quando in realtà essa cambia durante la trasmissione in maniera pseudocasuale, anche se è stato dimostrato in letteratura che è possibile condurre un attacco di sniffing senza particolari problemi anche in questo scenario [SPI, 2007].

BLE opera su un livello particolare chiamato GATT (General Attribute Profile) che definisce il sistema publish/subscribe che caratterizza questa versione di Blueooth [TOW, 2014]. Le entità presenti sono il Peripheral Device (il cosiddetto master, o server), entità che pubblica o presenta i dati, ed il central device (il cosiddetto slave, o client), entità che deve recuperarli. I dati vengono quindi esposti dal Peripheral device, pensati come contenitori di informazioni strutturare gerarchicamente attraverso i concetti di Servizi e Caratteristiche: in particolare, ogni Service è individuabile tramite uno UUID (128 bit per servizi custom) ed ognuno di essi contiene una o più caratteristiche, che sono le vere e proprie proprietà rappresentanti i dati rilevati: si veda figura 7 come riferimento. La comunicazione, quindi, avviene con la raccolta dei dati sulle proprietà, che è possibile leggere ed eventualmente scrivere, sia con un approccio a polling che sottoscrivendosi ad esse.

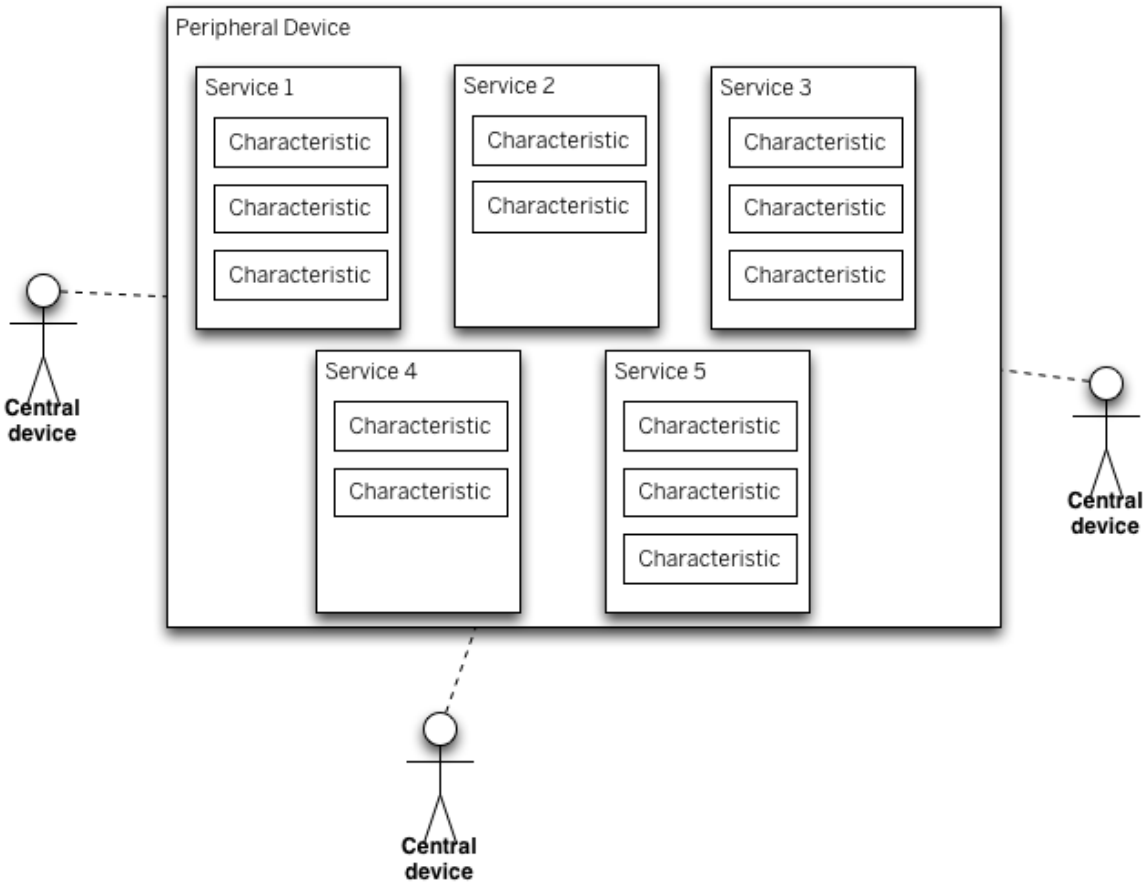

Figura 7: Interazione Central / Peripheral Device (CurieBLE library)[CBL]

La fase più delicata è ovviamente l'accoppiamento (*pairing*) tra i dispositivi Peripheral e Central, che è la fase dove viene anche gestito un eventuale supporto di sicurezza con lo scambio di alcune informazioni: è proprio in questa fase che Bluetooth e in particolare BLE definiscono, oltre alla metodologia di frequency hopping da adottare nella comunicazione, differenti livelli di sicurezza possibili, proponendo il cosiddetto *Secure Simple Pairing* [TRI, 2014].

2.2.2: Supporto alla sicurezza: Secure Simple Pairing (SSP)

Nello standard bluetooth vengono definiti quattro livelli di sicurezza:

1. Nessuna sicurezza (no autenticazione, no confidenzialità);
2. Pairing non autenticato con confidenzialità;
3. Pairing autenticato con confidenzialità;
4. *Secure Connection* autenticato con confidenzialità [REN, 2017];

Trascurando il livello 4 (dove viene usato l'accordo di Diffie-Hellman per la distribuzione di chiavi, considerato un protocollo troppo oneroso in termini di risorse computazionali per dispositivi a basso consumo di energia), si va a definire qual è il supporto pratico nell'implementazione di una di queste modalità di sicurezza [SIG].

La fase di Pairing tra dispositivo central e peripheral segue solitamente un flusso di questo tipo:

1. Fase di scanning: il dispositivo centrale svolge l'azione di scan alla ricerca di dispositivi peripheral limitrofi;
2. Fase di advertisement: il dispositivo peripheral, quando è connesso, manda dei messaggi di advertise al fine di essere visibile a chiunque voglia stabilire la connessione con lui;
3. Non appena l'advertise viene ricevuto dal dispositivo centrale, esso conclude la fase di scan e cerca di connettersi con il dispositivo periferico, una volta ottenuto l'indirizzo MAC;
4. Se tutto va per il meglio, il dispositivo centrale può iniziare a comunicare (leggere / scrivere caratteristiche)

È proprio in questa fase di negoziazione della connessione che BLE, oltre che gestire le caratteristiche del frequency hopping, offre diversi livelli configurabili di sicurezza, a seconda di come questo scambio di messaggi iniziale verrà poi effettivamente realizzato. In particolare, in base alla modalità di sicurezza richiesta, viene inserita una figura chiamata *Security Manager* (SM) per la trasmissione sicura dei dati, che va a ridefinire il processo di pairing appena descritto (Secure Simple Pairing), andando a distribuire delle chiavi tra le due entità [KWO et al. , 2016].

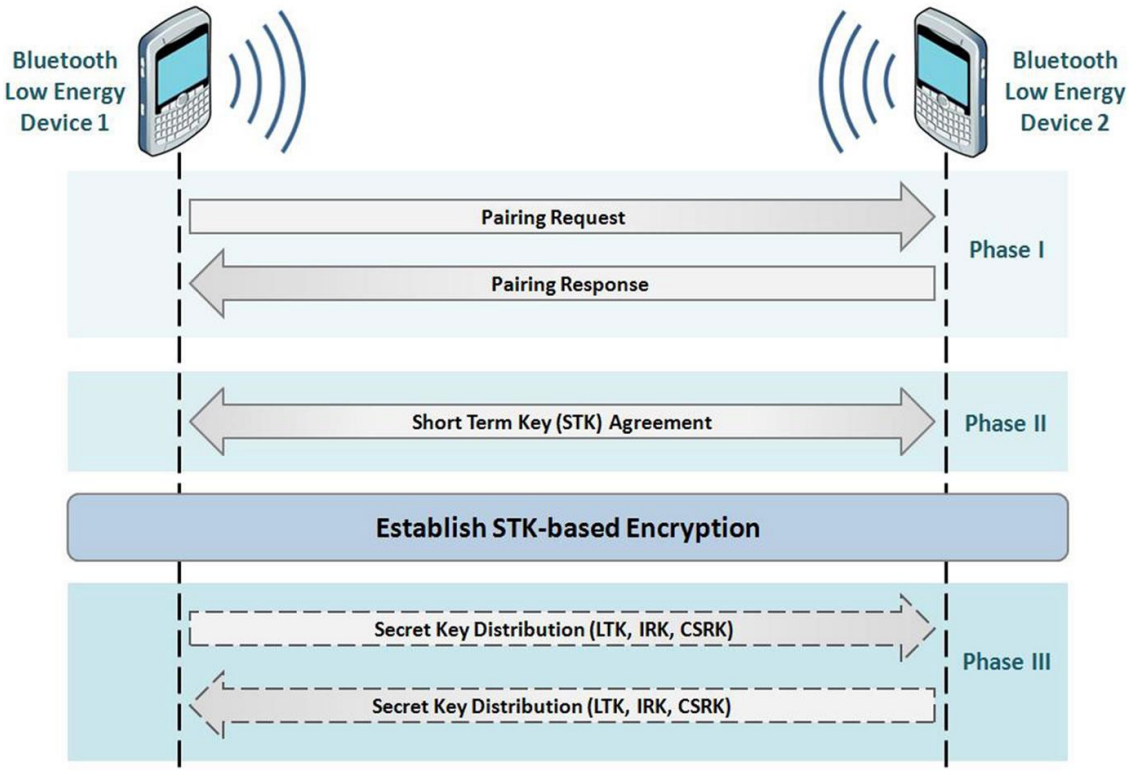

Figura 8: processo di Secure Simple Pairing

Nel dettaglio, la procedura di sicurezza è suddivisa in tre fasi, come si vede in figura 8: nella prima fase viene scambiato tra master e slave il messaggio di "pairing", composto da informazioni quali le capacità I/O, la massima lunghezza della chiave di crittazione e le frequenze dell'hopping da stabilire; nella successiva seconda fase viene accordata una Short Term Key (STK) e in un'ultima fase viene stabilita una Long Term Key (LTK), costruita attraverso la STK calcolata precedentemente. In particolare, alla fine della prima fase master e slave si accordano su quale sarà il metodo con il quale essi si scambieranno una ulteriore chiave che poi permetterà di generare la STK, la cosiddetta Temporary Key (TK): questo passaggio è fondamentale perché, una volta ottenuta TK, si andrà a generare un messaggio per la costruzione di STK. Il metodo per generare TK viene scelto esclusivamente in base alle capacità I/O degli attori della comunicazione:

- **Just Works (JW):**

Questo è il metodo scelto quando abbiamo un dispositivo che non ha possibilità di inserire input (cuffie bluetooth, ad esempio). Essa non garantisce alcun livello di protezione, dato che TK=0 (sono possibili ascolto passivo e MitM).

- **Passkey Entry:**

Questo metodo necessita dell'inserimento di sei cifre in un dispositivo capace di inserire input, e l'eventuale stampa e poi inserimento di queste cifre in un altro dispositivo che sia capace di output/input. Essa garantisce poca protezione (è possibile forza bruta).

- **Out Of Band(Oob):**

Questo metodo prevede l'utilizzo di un'altra interfaccia rispetto a quella bluetooth per scambiare informazioni sul pairing. Garantisce un ottimo supporto alla sicurezza. (es. interfaccia NFC)

Responder	**DisplayOnly**	**Display Yes/No**	**KeyboardOnly**	**NoInputNoOutput**	**KeyboardDisplay**
DisplayOnly	Just Works	Just Works	Passkey	Just Works	Passkey
Display YesNo	Just Works	Just Works	Passkey	JustWorks	Passkey
KeyboardOnly	Passkey	Passkey	Passkey	Just Works	Passkey
NoInputNoOutput	Just Works	Just Works	Just Works	Just Works	Just Works
KeyboardDisplay	Passkey	Passkey	Passkey	JustWorks	JustWorks

Tabella 3: relazioni capacità I/O – modalità di generazione TK [BON, 2015]

Riassumendo, in tabella 3 ci sono tutte le possibili configurazioni in base all'input dei due nodi Bluetooth.

Dopo aver scelto il metodo da utilizzare per la generazione di TK, viene scambiato un messaggio di MConfirm e SConfirm rispettivamente da Master e Slave per controllare che entrambi stiano utilizzando la stessa TK, ed infine con lo scambio di un numero casuale (MRand e SRand) viene controllato che chi manda la conferma è proprio chi dice di essere (autenticazione e identificazione); solo a questo punto ognuna delle due parti genera la STK. Lo slave infine decide una LTK, e mandandola crittata con STK verso il master si assicura che LTK rimanga segreta, e che le comunicazioni successive siano crittate e autenticate. La complessità di questo protocollo garantisce autenticazione e confidenzialità di tutte le comunicazioni successive (se, ovviamente, la prima fase non è stata intercettata da avversari, assumendo quindi che avvenga al sicuro). Il metodo OOB garantisce una sicurezza molto alta, ma ovviamente nella norma non è progettabile data la quasi impossibilità di sensori a basso costo e basso consumo di energia di avere più interfacce a livello di comunicazione per gestire questo metodo. Inoltre risulta subito un po' limitante il fatto di poter configurare sicurezza solamente quando abbiamo possibilità di inserire input o mostrare output con un dispositivo: la sicurezza non dovrebbe (e non deve) dipendere dalle capacità di un sensore, ma dai requisiti che vogliamo garantire.

Nonostante l'importanza di avere un supporto di questo tipo, la realtà dei fatti è un'altra: la maggior parte dei dispositivi IoT che utilizzano BLE messi in commercio manca completamente di questo supporto, e nel caso che il dispositivo non abbia capacità di input/output risulta molto semplice

attaccare la comunicazione e manometterlo. Essendo inoltre un protocollo di comunicazione di livello fisico, è pressoché impossibile riuscire ad avere in mano una modalità lato sviluppo di configurare queste lacune, se non provvedendo a creare *on-the-top* una qualsiasi regola applicativa che permetta di bypassare questo problema. Il meccanismo di frequency hopping garantisce una prima forma di protezione contro attacchi passivi o MitM, ma come si andrà a mostrare nel prossimo paragrafo esistono molte soluzioni open-source che riescono senza problemi a violare le comunicazioni bluetooth, ed eventualmente condurre attacchi mirati a questi dispositivi insicuri. Anche con l'utilizzo corretto di SSP è stata verificata la vulnerabilità della comunicazione bluetooth: infatti, il tool *crackle* [RYA, 2013] è uno degli esempi più famosi di come potrebbero essere sfruttate le vulnerabilità di questo protocollo.

Ogni protocollo può avere delle vulnerabilità, anche quello considerato più sicuro (eclatante l'esempio di WPA2 [CRT, 2017]), ma è importante in ogni caso avere la visibilità e la possibilità di modificare questi strumenti, che vivendo invece a livello fisico diventano molto difficile gestire attraverso un'azione di cross-layering sullo stack ISO/OSI. Urge uno strumento in grado di verificare in maniera automatica lacune di questo tipo: l'analisi dei dati e gli algoritmi di machine learning stanno andando proprio su questa direzione.

2.2.3: Analisi e progettazione per verificare vulnerabilità

Verificare le vulnerabilità dell'adozione della tecnologia BLE in un prototipo di sistema software IoT (e quindi nella semplice interazione *one-to-one* tra peripheral e central device) diventa a questo punto fondamentale per poter poi effettivamente rendersi conto se lo scenario di sicurezza da proteggere è "difeso" dalla progettazione del sistema stesso oppure no, sviluppare una contromisura applicativa per eventualmente garantire la sicurezza di questo scenario e collegare alla verifica una qualsiasi modalità automatica, per poi evitare nel futuro di andare a rifare questo controllo a mano. Si decide quindi di adottare uno stile pienamente Owasp (o STRIDE) per andare a delineare un sistema di sicurezza in relazione allo specifico attacco da verificare (o, dualmente, in relazione al requisito di sicurezza da proteggere). Questa metodologia di lavoro garantisce il beneficio che ogni sistema di sicurezza che verrà inserito nel sistema IoT, e che introdurrà chiaramente un qualche tipo di overhead, sarà comunque giustificato in relazione allo scenario da proteggere, che potrebbe essere delineato a tavolino dal committente o product owner: partire dall'attacco garantisce globalmente un'adozione più consapevole di un sistema di sicurezza prototipale (verifica, contromisura, validazione), che nell'estensione dell'architettura potrebbe garantire comunque la verifica della non violabilità dello scenario, in un'ottica pienamente guidata da test automatici.

A questo punto occorre ragionare ed analizzare come realizzare una verifica di questo tipo, e quindi come andare a progettare uno scenario di attacco all'interno della tecnologia BLE, tenendo in considerazione la sua natura. In realtà esistono diversi strumenti per realizzare un attacco: la scelta verterà su alcuni punti fondamentali, come la possibilità di collegare alla verifica di un particolare scenario uno o più test applicativi, la relativa riusabilità di questa verifica e soprattutto la possibilità di effettuarla con meno problematiche (e supporti) possibili. Lo scenario da verificare ovviamente dipenderà dalle esigenze applicative del sistema, ma, in generale, essendo BLE (e più in generale

Bluetooth) un protocollo soggetto ad attacchi di sniffing o comunque di intercettazioni, uno scenario classico potrebbe essere quello di andare a verificare che un terzo elemento malevolo riesca (o meno) ad intromettersi nelle comunicazioni tra le entità peripheral e central autorizzate, nel classico scenario MitM (Man in the Middle). Le soluzioni proposte in letteratura e open-source riguardano diverse modalità con il quale poter realizzare una verifica di questo tipo: una prima modalità, che non dipende dal protocollo bluetooth in questione, potrebbe consistere nell'attacco di relay [FRA et al. , 2010], dove viene effettuato in alcuni esempi un attacco di relay nell'apertura smart di una macchina key-less. In questa tipologia di attacco, nonostante venga visualizzata molto bene la vulnerabilità ed elencate le possibili conseguenze critiche dell'attacco, viene messo in risalto come comunque ci sia un bisogno forte di attrezzature ad-hoc (antenne, ripetitori), per realizzare un sistema capace di visualizzare queste vulnerabilità critiche. Questi strumenti vengono usati proprio per fare sniffing di rete: per il BLE esiste addirittura un vero e progetto open-source Ubertooth [GIT1], descritto come piattaforma wireless capace di fare sniffing di rete su una comunicazione di rete.

Senza andare a visualizzare nello specifico il funzionamento di Ubertooth o di altri sniffer fisici per il bluetooth, fare sniffing di rete ed analizzare i pacchetti presenti nelle comunicazioni potrebbe essere in generale una buona soluzione per progettare poi su questi pacchetti intercettati alcuni scenari di attacco, come la raccolta di informazioni sensibili o addirittura l'implementazione di un attacco classico MitM tra le entità. Esistono innumerevoli tool che permettono di svolgere abbastanza facilmente questo lavoro: tra gli altri, Wireshark [WIR], tool che permette di catturare qualsiasi tipo di traffico rete e svolgere funzioni anche particolari sui pacchetti intercettati, tramite diverse librerie e binding con tecnologie di sviluppo (es. Python : pyshark[GIT2] , Java: jpcap[GIT3]): l'importanza di avere bindings verte soprattutto sul fatto che in questo modo è possibile implementare dei veri e propri test automatici. In molti riferimenti [ARG, 2013] sono evidenziate le diverse modalità e tecniche nello sniffing di pacchetti BLE, e viene messo anche in evidenza un problema molto importante: la presenza del frequency hopping rende molto difficile per un attaccante effettuare un'intercettazione pulita della comunicazione. Infatti [JAM, 2017] solo con specifici strumenti sofisticati è possibile disturbare o comunque intercettare la comunicazione bluetooth rispetto ad esempio quella Wi-Fi (che utilizza canali fissi), proprio a causa dell'utilizzo di Frequency Hopping da parte del bluetooth, che rende molto più complicato lo sniffing rispetto alla comunicazione Wi-Fi [EEN, 2014].

Si passano al vaglio anche oltre opzioni: nell'idea di volersi appoggiare su vere e proprie infrastrutture di test IoT in grado di provare a fare test anche "fisici" sul dispositivo (IoT) senza andare a delineare azioni di sicurezza come test di penetrazione a basso livello (siamo a livello prototipale e non ad architettura già sviluppata), è possibile utilizzare veri e propri applicativi o framework che permettano di creare una potenziale infrastruttura di test (sugli scenari malevoli). In particolare, tra le soluzioni open-source, si è deciso di tenere in considerazione quali potrebbero essere gli strumenti che rispecchiano maggiormente i criteri stabiliti a inizio progettazione, ovvero relativa semplicità di utilizzo, potenziale estendibilità e collegamento con tecnologie (linguaggi) di sviluppo, ad esempio con l'utilizzo di librerie. Ed ecco che le soluzioni presenti nella tecnologia BLE sono essenzialmente due: framework **BtleJuice** [CAQ1, 2016] [CAQ2,2016] e framework **Gattacker** [JAS1, 2016] [JAS2, 2016].

Entrambe le soluzioni, sviluppate per ambiente Unix negli ultimissimi anni come progetti open-source, entrambe scritte in NodeJS e con binding verso Python, propongono una valida alternativa all'infelice operazione di sniffing di pacchetti BLE con l'utilizzo di hardware dedicato e sofisticato, con le varie difficoltà dovute al frequency hopping e all'instabilità della rete: essi si prefiggono l'obiettivo di andare a costruire un'infrastruttura in grado di impersonare fisicamente l'attacco MitM, andando ad agire sulla fase di pairing. Le soluzioni sono molto simili tra loro, in quanto BTLEJuice utilizza proprio l'infrastruttura realizzata da Gattacker per poter offrire il proprio prodotto attraverso un'interfaccia web: nella fase di scan, il framework (che prende "le sembianze" del vero peripheral device) realizza un'azione di fase di advertisment più "frequente" rispetto a quella del dispositivo peripheral legittimo, costringendo il central a collegarsi con lui anziché col vero peripheral. A questo punto, il framework si è posto in mezzo rispetto ai due lati della comunicazione legittima, senza che i due attori se ne siano accorti, e tutte le informazioni (letture/scritture di dati) passeranno per lui, che può implementare logica applicativa ed effettivamente testare qualsiasi scenario malevolo.

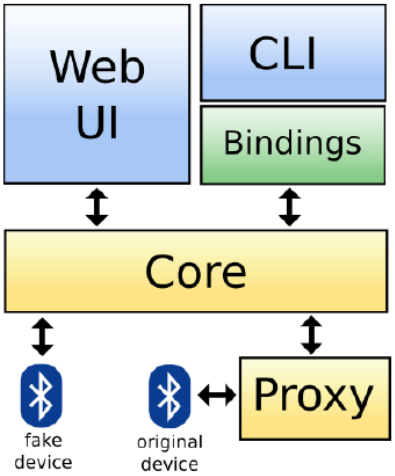

Figura 9: Architettura BTLEJuice

Nel dettaglio, il framework **BtleJuice** mostrato in figura 9 è composto da due elementi, *interception proxy* e *interception core*, che devono andare in esecuzione su due diverse macchine: il proxy in una prima fase si collegherà al peripheral originale prendendone le sembianze, sia agli occhi del central legittimo, sia agli occhi dell'interception core che si collegherà via web ad esso. Una volta associato via web il core al proxy, attraverso l'interfaccia web si vedranno le varie operazioni che il central esterno andrà a svolgere sul dispositivo (connessione, lettura e scrittura) che crede sia il vero peripheral, ma che in realtà è il proxy. Ed è dal core che è possibile andare a recuperare queste operazioni per associarci una logica applicativa, sia via NodeJS o Python, sia attraverso l'utilizzo dei pulsanti dell'interfaccia web (modifica dei dati *on-the-fly*, per esempio).

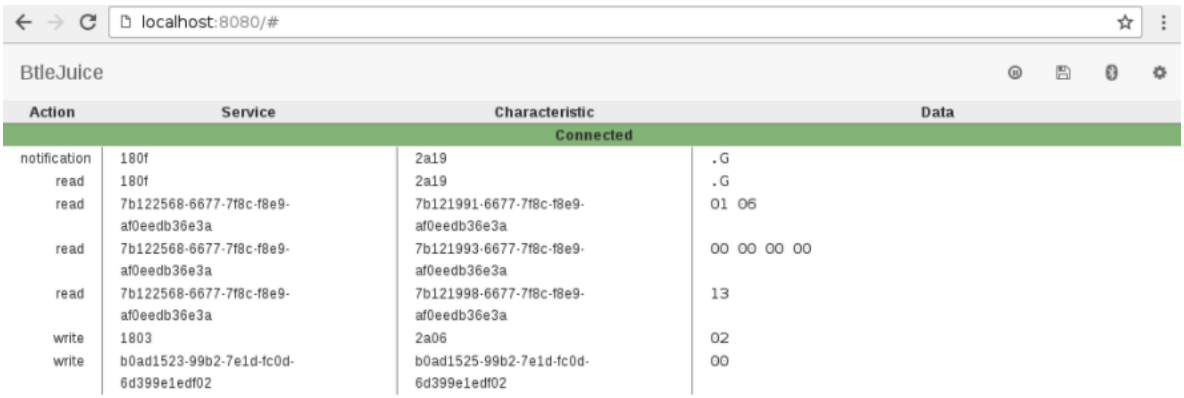

Figura 10: Interfaccia Web BTLEJuice

Per quanto sia un punto decisamente a suo favore l'utilizzo di un'interfaccia web per rendere più semplice il flusso delle operazioni da testare, come si vede nella figura 10, occorre considerare che la scarsa documentazione, il poco utilizzo e i continui problemi di rete legati al suo utilizzo a causa di uno stato ancora troppo "immaturo" della soluzione stessa, sono problemi importanti, soprattutto nell'ottica di dover costruire test robusti che riescano a essere decisivi anche in uno scenario più complesso di un'interazione semplice *one-to-one*.

Gattacker [JAS3, 2016] rappresenta una soluzione un po' più complessa architetturalmente ma più utilizzata perché maggiormente stabile: utilizza la stessa tecnica di Btlejuice, realizzando un attacco MitM attraverso la clonazione del vero peripheral device grazie al framework stesso, e lo fa utilizzando due macchine che fungono da proxy ai dispositivi legittimi (central proxy e peripheral proxy), costruendo l'infrastruttura sulla quale possono essere testati diversi scenari sulle comunicazioni senza che i dispositivi central e peripheral legittimi se ne rendano conto in alcun modo: si veda in figura 11 l'architettura.

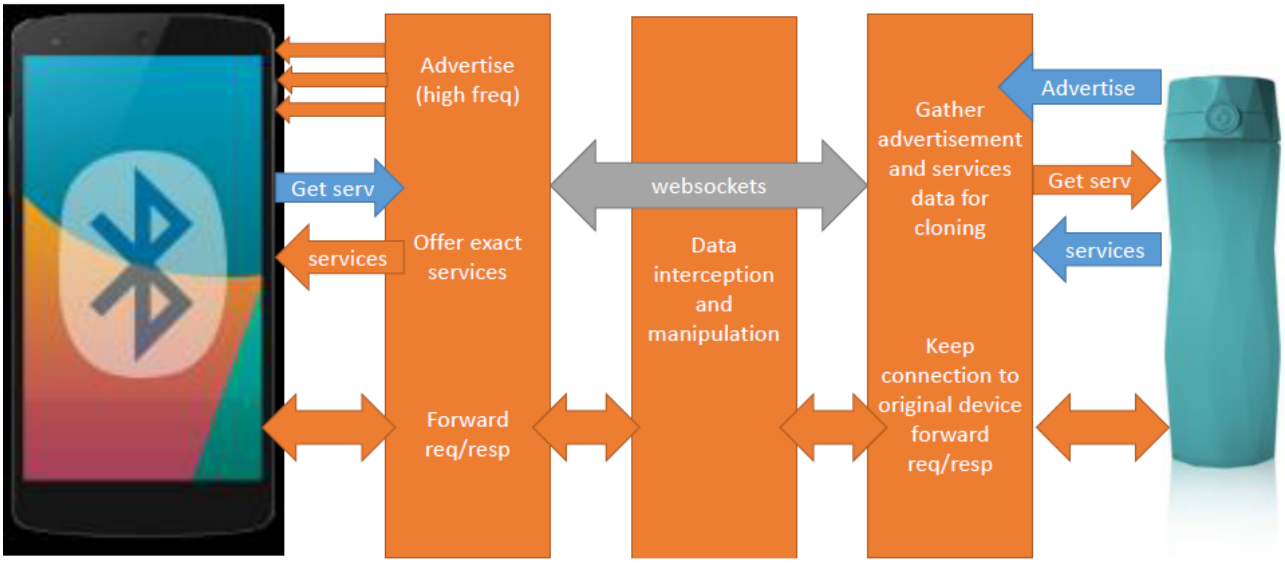

Figura 11: Presentazione Gattacker @ BlackHat 2016.

Da sinistra verso destra: Central Device (autorizzato) / Central Proxy / Logica applicativa / Peripheral Proxy / Peripheral Device

È fondamentale comprendere come venga clonato il dispositivo e qual è il ruolo del componente che viene definito come "Logica Applicativa", oltre alle fasi di setup e di connessione che garantiscono che l'infrastruttura venga creata. Il peripheral proxy clona il dispositivo sostanzialmente andando a copiare dentro di sé (in un file json, figura 12) la gerarchia di servizi e caratteristiche, i contenitori dei dati del peripheral device BLE, e permette sulle singole caratteristiche di questa gerarchia di definire delle vere e proprie funzioni d'intercettazione (*hooks function)* che permettono di andare a chiamare a partire da questo file dei metodi scritti in file NodeJS, sulla quale è possibile definire funzioni in grado di implementare una qualsiasi logica applicativa richiesta.


```
{
    "uuid": "06d1e5e779ad4a718faa373789f7d93c",
    "name": null,
    "properties": [
        "write",
        "notify"
    ],
    "startHandle": 8,
    "valueHandle": 9,
    "endHandle": 10,
    "descriptors": [
        {
            "handle": 10,
            "uuid": "2902",
            "value": ""
        }
    ],
    "hooks": {
        "dynamicWrite": "dynamicWriteFunction",
        "dynamicNotify": "customLog"
    }
}
```


Figura 12: Gerarchia file json gattacker con hook sulla caratteristica

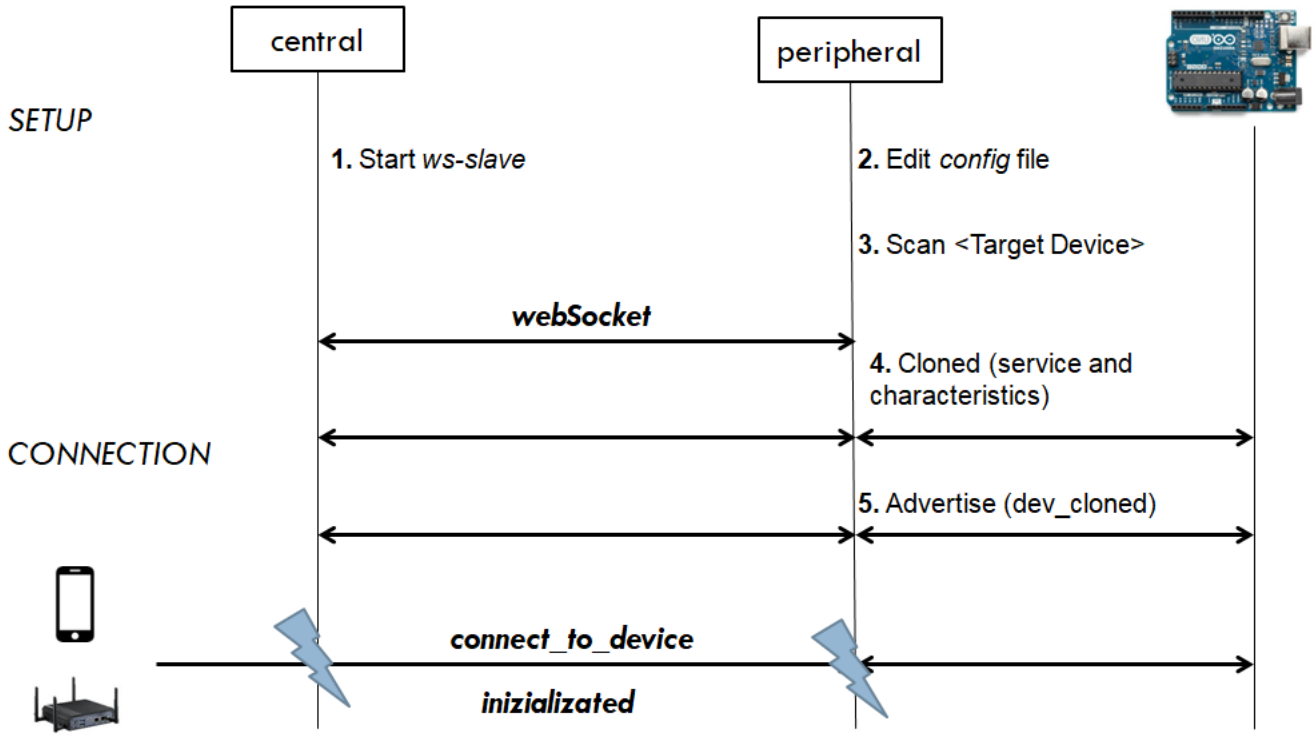

Figura 13: configurazione framework gattacker

Andando maggiormente nel dettaglio con la figura 13, le fasi di setup e di connessione tra i diversi elementi dell'architettura costituiscono il nucleo dell'intero framework. In una prima fase (**1.)** il dispositivo central proxy resta in ascolto di una connessione WebSocket che verrà poi effettuata dal peripheral proxy, il quale contiene un file di configurazione interno da modificare **(2.)** ed è in grado inoltre di svolgere la funzione *scan* **(3.)**, andando a connettersi al peripheral device e a copiarsi i suoi dati. Una volta connessi central e peripheral proxy tra loro via WebSocket, il peripheral proxy, che ha clonato i dati del peripheral device **(4.)**, contiene al suo interno anche la funzione di *advertise*, andando essenzialmente a sostituirsi al vero peripheral device (**5.** , con la possibilità di clonare pure il suo indirizzo MAC) e a costringere un qualsiasi dispositivo central che voglia connettersi al vero peripheral device a connettersi in realtà con il central proxy **(6.)**. E' su questa infrastruttura poi che verranno definiti tutte le varie operazioni applicative che è possibile svolgere con i dati (eventualmente) intercettati.

Gattacker, nonostante la complessità architetturale, risulta oggi essere un framework molto più stabile rispetto a BtleJuice: anche se sprovvisto di una vera interfaccia Web per svolgere le operazioni, ha la possibilità di configurare via codice (json) elementi di logica applicativa, risultando più vicino allo sviluppo e alla definizione di operazioni "general-purpose". Per la connessione al vero peripheral device, è definita come accennato una funzione di clonazione dell'indirizzo MAC che però è una funzione gestita dallo stack Bluez [BLU, 2000] ed è supportata solo da alcune case produttrici della scheda bluetooth: nel caso di una prova sullo scenario reale, questo potrebbe risultare limitante, a meno di avere degli adattatori ad-hoc [GIT4, 2017].

SCENARI	PRO (V)	CONTRO (X)
Sniffer (es. WireShark)	Possibilità di manipolazione diretta di pacchetti di rete. Possibilità di binding tecnologici per implementazione logica applicativa. Larga documentazione e bacino di utenti	Utilizzo di costosi dispositivi fisici (antenne, sensori, adattatori) per uno sniffing accurato. Difficoltà di cattura e disturbo della comunicazione bluetooth a causa della sua natura (Frequency Hopping)
Framework #1 (BtleJuice)	Possibilità di effettuare tramite librerie direttamente l'attacco in piena ottica di TDD. Utilizzo semplificato attraverso interfaccia Web ed architettura essenziale	Scarsa documentazione e piccolo bacino di utenti. Instabilità elevata anche nello scenario ristretto.
Framework #2 (Gattacker)	Possibilità di effettuare direttamente l'attacco in piena ottica TDD. Buona documentazione e ottima stabilità in scenario ristretto di utilizzo.	Architettura complessa e necessità di adattatore per implementazione attacco sul dispositivo fisico.

Tabella 4: Analisi su soluzioni per ricerca vulnerabilità BLE

2.2.4: Analisi e progettazione per contromisure

Dopo aver dettagliatamente documentato quali sono i punti chiave nella ricerca di una metodologia per verificare vulnerabilità all'interno di un'interazione BLE (e motivato perché è necessaria questo tipo di analisi all'interno del flusso di lavoro), è necessario analizzare quali sono i punti chiave nella progettazione di un'eventuale contromisura da implementare. Non solo: una volta progettata la contromisura, è essenziale progettare un modo per poter validare questa contromisura nello scenario applicativo di riferimento, per poi passare all'eventuale iterazione successiva. Come ampiamente descritto, BLE propone diversi supporti per quanto riguarda la sicurezza, ma come specificato questi supporti non solo risultano essere non modificabili lato sviluppo, ma essi molto spesso non vengono adottati dalle case produttrici e, se adottati, risentono di alcune vulnerabilità critiche su cui è impossibile fare pienamente affidamento. Inoltre, nella definizione di uno scenario malevolo, è molto complicato andare a controllare se effettivamente una delle modalità che BLE mette a disposizione in maniera automatica riesce a invalidare questo scenario oppure no: le uniche piene modalità di sicurezza garantite da BLE (*Secure Connection* e l'utilizzo del metodo *OOB* per la generazione della chiave *TK*) risultano essere senza dubbio troppo onerose per sensori IoT BLE che comunicano a basso consumo di energia e bassa potenza computazionale.

In questo scenario di difficoltà di utilizzo del supporto di sicurezza BLE e della complessità di effettuare dunque operazioni di cross-layering di sviluppo per andare a configurare questi supporti di sicurezza, potrebbe essere una valida soluzione andare a definire delle modalità e dei protocolli

sul layer applicativo, in maniera da poter configurare pienamente lato sviluppo in maniera autonoma come proteggere particolari scenari. La difficoltà più grande rimane ancora quella della bassa capacità computazionale dei sensori IoT, ma ci sono discussioni aperte [GRA et al. , 2015] ed esempi reali e concreti[SHA et al. , 2017] che vanno proprio ad implementare contromisure per quanto riguarda rispettivamente crittografia ed autenticazione, modificando ovviamente i meccanismi classici in maniera che possano essere implementati nel contesto IoT.

Per quanto riguarda la validazione di una specifica contromisura, quello che deve accadere in ottica TDD è che i test di sicurezza che nella fase precedente hanno messo in risalto la vulnerabilità vengano effettivamente riprovati con la contromisura e diano un esito positivo (ovvero, verificare che la contromisura abbia funzionato). Questo passaggio non è banale, perché molte volte non è possibile riutilizzare le stesse tecnologie che hanno verificato la vulnerabilità. Ad esempio: se è stato usato un framework open-source per verificare uno scenario di MitM, andare a provare gli stessi framework (Gattacker e BtleJuice) dopo aver implementato la contromisura non avrebbe senso perché entrambi i framework vanno a clonare il dispositivo in una iniziale fase di *scan*, non modificabile, rendendo impossibile la verifica della contromisura così com'è. Per questo motivo è possibile considerare diverse opzioni: lo strumento più utilizzato per verificare se effettive vulnerabilità siano sfruttabili è senz'altro l'utilizzo di tecniche di *penetration testing* [FOR, 2011]. Con penetration test vengono intese tutte quelle operazioni che vengono usate (molto spesso da analisti di sicurezza) con l'obiettivo di individuare, determinare e sfruttare particolari vulnerabilità su architetture software complesse e molte volte persino completamente sconosciute internamente. In questo scenario di lavoro, l'utilizzo dei penetration test potrebbe essere applicato ad una situazione di lavoro in cui l'architettura è perfettamente conosciuta (fase prototipale): in particolare, all'interno di una specifica distribuzione di Linux (chiamata Kali) sono presenti diversi tool per effettuare operazioni di penetrazione a seconda del dominio di riferimento (raccolta di informazioni, analisi delle vulnerabilità, attacchi wireless, tool per applicazioni web[KAL]). Con la progettazione di penetration test ad-hoc è possibile rientrare pienamente in ottica Owasp e TDD: potrebbe risultare problematico però il fatto di utilizzare un nuovo sistema operativo per effettuare queste operazioni (Kali), ma soprattutto la complessità dei tools potrebbe essere un livello troppo dettagliato per la semplice verifica di specifiche prototipali. In molti casi potrebbe essere invece molto interessante andare a validare una particolare contromisura con degli scenari reali di utilizzo dell'interazione BLE prima e dopo la sua progettazione: in particolare, in uno scenario di verifica MitM la verifica dell'effettiva possibilità di effettuare alcune operazioni "malevoli" prima e dopo l'inserimento di un protocollo applicativo di contromisura potrebbe essere un test, anche se abbastanza semplice, molto interessante per validare lo scenario, soprattutto se tutta questa operazione fosse automatizzata per eventuali modifiche future. Occorre in ogni caso stabilire scenari reali abbastanza significativi, e talvolta affidarsi anche a una quantità di dati abbastanza considerevole per effettuare delle prove reali; si riassumono i pro e i contro nella tabella 5.

SCENARI DI VALIDAZIONE	PRO (V)	CONTRO (X)
Penetration Testing (Kali Linux)	Il modo più utilizzato e più completo per verificare vulnerabilità sia a livello di architettura sia a livello di codice: pienamente integrato con la metodologia TDD.	Inesistenza di tool che verifichino condizioni prototipali, per quanto riguarda manomissione o MitM di sensori non connessi in rete (WEB)

	Un attacco completo allo stack ISO/OSI può essere condotto, implementato, e quindi verificato	
Scenari Reali	Prova effettiva della contromisura su casi reali di interazione possibile tra i dispositivi. Semplicità dell'inserimento di queste operazioni di prova all'interno del codice di sviluppo e dei test di sicurezza.	Complicata progettazione di tutti gli scenari reali possibili di validazione della contromisura. Rimanendo a livello applicativo (e non di rete), difficile avere uno strumento che verifichi un effettivo attacco completo allo stack ISO/OSI

Tabella 5: Differenti scenari di validazione per contromisura

2.3 Metodologie Agili

L'intera storia dell'ingegneria del Software è stata scritta nel tempo dallo sviluppo di differenti metodologie di produzione del Software stesso. Nel 1956, per la prima volta, Herbert D. Benington presentò una metodologia di sviluppo di un prodotto che riprendeva pienamente i principi di un metodo che, a partire dal 1970, fu denominato *waterfall model,* il cosiddetto modello a cascata per la produzione del software [HUG, 2009], come illustrato in figura 14.

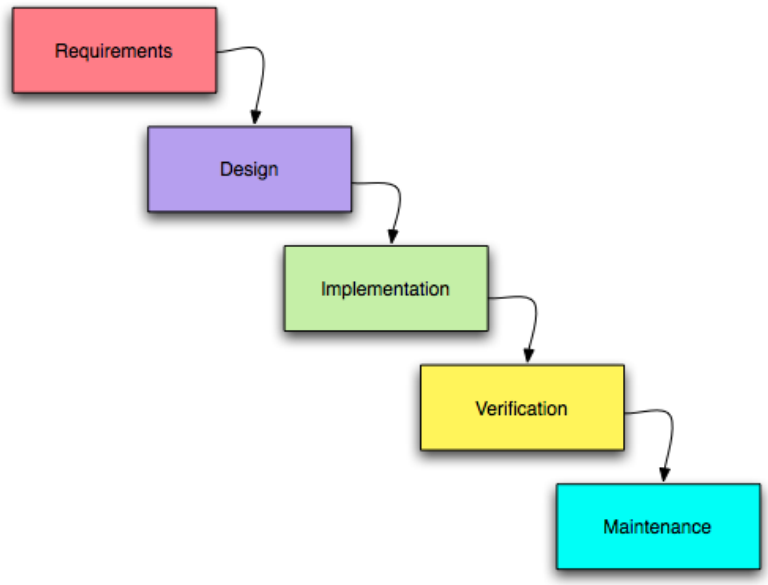

Figura 14: Modello a cascata per la produzione del Software

In questo modello, nella produzione di un sistema software sono presenti delle fasi fisse che vengono eseguite obbligatoriamente in ordine: produzione di un documento dei requisiti relativi al sistema software, analisi di questi requisiti, progettazione dell'architettura software che va a modellare le funzionalità del sistema, implementazione, integrazione e operazioni di testing dell'intera applicazione, deploy e infine manutenzione dell'intero sistema. Il passaggio tra queste diverse fasi avviene solamente quando la fase precedente è terminata, molte volte a seguito della produzione di una documentazione associata. Con questa metodologia di lavoro l'obiettivo è quello innanzitutto di ridurre i costi di un eventuale problema che potrebbe essere riscontrato in fase "matura" del software, cercando di gestirlo anticipatamente con una fase dettagliata di analisi dei requisiti, spendendo meno per la sua risoluzione. Inoltre, questa metodologia pone la stessa importanza sia alla documentazione che al codice sorgente, documentazione vista come rappresentazione della conoscenza che deve essere tramandata all'interno di un team di lavoro. Questo modello, per quanto sia stato l'unico usato per alcuni decenni, e nonostante sia ancora attualmente adottato da qualche realtà IT, nel corso degli anni ha subito molte critiche che, seppur graduali, hanno portato alla ribalta una nuova metodologia di lavoro, basata su principi completamente diversi e in antitesi con i principi *waterfall*. I primi limiti di questa metodologia furono messi in risalto soprattutto con lo sviluppo tecnologico a partire dagli anni '90, dopo aver riscontrato che attraverso la metodologia di lavoro a cascata era impraticabile rispondere a modifiche di requisiti continue da parte di clienti che decidevano quali fossero le esigenze di un software solo dopo averlo toccato, visto e attraverso dei feedback di utilizzo, proprio negli anni in cui i sistemi software mutavano completamente (e il Web prendeva il sopravvento). Un eventuale cambio di requisiti non era contemplato nella metodologia a cascata che, non essendo iterativa, rispondeva a un cambiamento ripercorrendo tutte le fasi in ordine, tra cui la progettazione di un'eventuale nuova architettura da zero. Con l'utilizzo della metodologia *waterfall* il cambiamento veniva visto con un'accezione completamente negativa, quando invece diventava essenziale in quegli anni di trasformazione graduale dei sistemi software avere un approccio positivo e soprattutto reattivo a un cambiamento inaspettato, prevedendo tutto ciò attraverso una metodologia ben diversa da quella a cascata.

Questa esigenza venne tradotta nel 2001 in un vero e proprio Manifesto, presentando così al mondo dello sviluppo un'innovativa metodologia di lavoro per la produzione di un sistema software: la metodologia *Agile*.

2.3.1: Principi ed Ecosistema Agile

La metodologia di lavoro Agile risponde direttamente ai principi del manifesto [BEC et al. , 2001] che l'hanno introdotta nel mondo dello sviluppo:

1. Individui e interazioni più importanti di processi e tools;
2. Software funzionante più importante di una documentazione comprensiva;
3. Collaborazione con i clienti più importante della negoziazione di un contratto;
4. Rispondere ai cambiamenti più importante di seguire un piano lineare.

da "Agile Manifesto", 2001

Per quanto il vero valore economico stia nei termini a destra di ogni paragone, i fautori di questo manifesto vollero porre volutamente in evidenza i termini a sinistra, partendo da queste prime linee guida fino a proporre una vera e propria metodologia di lavoro basata su principi semplici, tra cui:

- Soddisfazione completa del cliente tramite rilasci continui;
- Accoglienza totale al cambiamento, vedendolo come sfida positiva da cogliere per valutare l'efficacia del processo utilizzato;
- Convivenza lavorativa progettuale costante tra sviluppatori e manager;
- Costruzione di progetti sulla motivazione di ogni individuo del team;
- Conversazione faccia a faccia per la risoluzione e confronto su ogni tipo di problema o richiesta;
- Team auto-organizzanti, capaci di sapersi gestire, confrontare, suddividere il lavoro per raggiungere risultati sempre migliori;
- Riflessione nel team a intervalli regolari su come diventare maggiormente efficienti, aggiustando a mano a mano comportamenti e scadenze

Da questi principi si evince un aspetto che è in comune a tutti: il valore dell'interazione tra le persone, che si gioca nella collaborazione all'interno del team, nel dialogo con il cliente, nel dialogo con manager o altre figure aziendali. Esso, più di ogni altro, da' importanza all'individuo, al cliente, al team e alla collaborazione tra questi ruoli rispetto al mero valore economico. Sebbene queste linee guida descrivano pienamente la natura della metodologia di lavoro Agile, esse non danno in realtà uno strumento tecnico con il quale metterle in pratica, né un processo strutturale che un'azienda può effettivamente utilizzare per poter usufruire dei benefici di questo approccio. Per questo motivo si parla di un vero e proprio ecosistema [HIG, 2002] Agile, dove al suo interno esistono principalmente tre mondi che sono accomunati dagli stessi principi, ma che presentano soluzioni e processi tecnologici differenti: *Scrum, Kanban* (Lean Development) ed *eXtreme Programming* (XP, d'ora in poi).

È possibile definire sia Scrum che Kanban [BOW, 2015] come veri e propri strumenti che danno a disposizione un sistema, un processo strutturato, un framework astratto di lavoro Agile, escludendosi però mutuamente tra di loro a causa delle reciproche differenze, mentre è possibile definire XP come un insieme di pratiche tecniche che vanno ad implementare i principi dell'Agile. Kanban [STE et al., 2014], in breve, è un metodo utilizzato soprattutto per progetti consolidati già in fase di produzione, ad esempio applicazioni legacy che non hanno un vero bisogno di nuove funzionalità aggiuntive bensì di un refactoring del processo di manutenzione, nell'ottica di far in modo che sia il team di sviluppo il responsabile della definizione delle suddivisioni delle attività. Non più quindi un manager o una persona esterna che definisce uno schema fisso, ma il team che con operazioni di *pull* si autogestisce nella schedulazione delle diverse attività, con l'idea che, vista la maggior conoscenza del progetto in corso, sia il team stesso a distribuirsi le risorse e a suddividere le attività autonomamente. Mentre questa metodologia di lavoro per un prodotto in fase di prototipo non è molto interessante, alcune pratiche dell'XP e soprattutto la metodologia di lavoro Scrum vengono ritenute più adatte da approfondire: si veda in figura 15 il ciclo classico di sviluppo Agile.

Figura 15: Rappresentazione schematica della metodologia di lavoro Agile

2.3.2: Extreme Programming e Test driven development

La metodologia di Extreme Programming (XP)[AGW] descrive delle regole pratiche per lo sviluppo software che vengono definite estreme proprio perché secondo la definizione proposta dal suo creatore *Kent Beck* (2000) esse dovrebbero essere applicate sempre nella produzione di software in contesto Agile, a prescindere dalla metodologia utilizzata all'interno del team (Scrum o Kanban). Le diverse pratiche di programmazione XP hanno la caratteristica comune di rimanere in linea con la filosofia Agile: valorizzare interazioni umane tra gli sviluppatori ed il cliente, valorizzare i feedback di questi ultimi e reagire positivamente ad un cambiamento, andando a proporre tecniche comuni di semplicità, chiarezza e pulizia del codice sorgente, scrivendo solamente ciò che è stretto necessario e andando ad aggiungere componenti secondari in fasi ulteriori di *refactoring.* In particolare, tra le tecniche XP più utilizzate si riscontrano:

- *Test Driven Development (o TDD):* scrivere i Test prima di iniziare a sviluppare la parte funzionale, andando a testare ogni singolo componente Software prima della sua integrazione con l'applicazione;
- *Pair Programming*: due sviluppatori lavorano insieme sullo sviluppo di funzionalità software, dove uno programma e l'altro osserva, a rotazione;
- *Continuos Integration*: a ogni funzionalità implementata si associa il deploy su di un server a cui sono associati test automatici e di integrazione, così che ogni componente del team può accedere in ogni momento a una versione funzionante del software;
- *Standard coding*: mantenere all'interno del team le stesse convenzioni nel codice, eliminando ambiguità;
- *Refactoring*: rifattorizzare e ridisegnare il software senza andare a cambiare le funzionalità e il comportamento, per renderlo più semplice e riusabile, eliminando eventuali *code smells.*

In generale, l'utilizzo di tutte queste tecniche o comunque della maggior parte di esse da' la possibilità a un team di sviluppo di operare in maniera Agile, ottenendo molteplici benefici e garantendo i principi chiave di questa metodologia, che ovviamente riprendono la filosofia del

Manifesto: comunicazione, semplicità, feedback, coraggio e rispetto. Andando più sul particolare, si vanno ad approfondire i benefici di TDD, inteso come modello di sviluppo software che prevede che la stesura di test unitari avvenga prima della scrittura di quella del software che deve essere sottoposto a test, come spiegato in figura 16.

Figura 16: Test Driven Development flow

In letteratura [MAR, 2008] spesso il codice viene descritto come il vero e unico dettaglio dei requisiti: anche se la direzione futura è sempre più quella di proporre linguaggi ad altissimo livello o tool automatici che riescano a tradurre quasi per magia requisiti di un cliente in maniera precisa, in qualche livello ci sarà sempre un codice, ed è importante che questo codice sia il più accurato, formale e comprensibile da una macchina: un codice essenzialmente "pulito". Tra le varie definizioni di codice pulito è presente in particolare il concetto che "il codice, per essere pulito, debba essere testato, in maniera tale che le cose vengano fatte in un solo modo". L'intenzione è quella di concretizzare attraverso tre leggi la definizione di una vera e propria pratica che verrà chiamata con il nome di Test Driven Development (TDD), basata essenzialmente sulla scrittura di Unit Tests (intesi come test che verifichino il comportamento di un piccolo componente software non ancora integrato con altri), e utilizzata pienamente nel contesto di Metodologie Agili e pratiche di XP:

- Non dovresti scrivere codice di produzione finchè non hai scritto un unit test associato a esso che fallisce;
- Non dovresti scrivere più di un unit test di ciò che è sufficiente a farlo fallire;
- Non dovresti scrivere più codice di produzione rispetto a quello sufficiente per fare passare i test unitari.

In figura 16 vengono tradotte le diverse leggi del TDD nel flusso applicato normalmente nella produzione di componenti software. Il più grande beneficio dell'utilizzo di questa metodologia di lavoro è che in questo modo il codice di produzione sarà sempre pulito e accurato, dato che nel momento in cui il software nasce attraverso la verifica di aspetti comportamentali, eventuali modifiche che cambiano il comportamento di questo sistema software vengono gestite "by design" dai test: migliorare l'architettura del codice, eliminare ridondanze o *boilerplate code,* sono operazioni che non avranno ripercussioni sul comportamento del software, data la presenza sicura e costante di una suite di test che ha definito il software stesso.

La chiave per mantenere un'architettura e un codice pulito e accurato sta proprio nella progettazione della suite automatica di test unitari che garantiscono il corretto comportamento dei componenti software: saranno poi i test in fase di integrazione (*Integration Test*) a testare end-to-end i diversi componenti in collegamento tra di loro, ma in ogni caso essi non solo non saranno complicati da progettare data la presenza di una suite di test unitari già assodata, ma questa suite stessa permette una maggiore sicurezza sulla non violabilità del comportamento dei componenti, anche quando l'architettura diventa molto complicata. Flessibilità, manutenibilità, scalabilità, riusabilità del codice di produzione, tutto passa necessariamente dalla scrittura di test (che devono essere, a loro volta, accurati e puliti).

L'intero sistema di produzione di software Agile fa spesso poi riferimento nel dettaglio a un complesso e lungo processo automatizzato all'interno del team, dove questi Test sono elementi centrali: la cosiddetta pipeline di processo chiamata *Continuos Delivery* [SCA, 2010].
La pipeline di Continuos Delivery (chiamata anche semplicemente Pipeline) rappresenta le attività, i workflow, il processo di automazione che permette un rilascio continuo di valore per l'utente finale (quindi la concretizzazione di uno dei principi Agile). Se consideriamo un intero processo Agile, esso mantiene e condivide una pipeline con gli aspetti e le tecnologie necessarie per portare il valore della soluzione nella maniera più indipendente e automatica possibile: tutti gli elementi di questa pipeline lavorano insieme per supportare la consegna di ogni piccola modifica o di una nuova funzionalità, che sono rilasciate in accordo alla domanda di mercato (o del cliente specifico).
Ecco dunque che con l'infrastruttura costruita dall'utilizzo di questa pipeline si riescono a concretizzare pienamente i valori espressi dal manifesto, come rilascio continuo e reazione positiva al cambiamento: ogni fase, dominata in gran parte da test automatici sui componenti e sulla loro integrazione, produce feedback e risultati che vengono continuamente e in tempo reale valutati da chi sviluppa codice, con lo scopo di controllare che l'intero codice e processo sia corretto, e in maniera automatica fare in modo che l'utente abbia a disposizione velocemente la nuova funzionalità senza doversi preoccupare di ricostruire l'infrastruttura di deploy. Manutenibilità, estendibilità, riusabilità di un sistema anche complesso diventano un compito abbastanza semplice con l'utilizzo di questa metodologia, che a sua volta nella complessità e diversità delle tecnologie che utilizza ad ogni livello riesce a dare un enorme supporto all'ecosistema di produzione Agile del software. L'importanza dei test automatici, infine, assicura che il rilascio del software avvenga solo nel momento in cui queste suite verificano su di esso ogni aspetto desiderato, come mostrato in figura 18, e una modifica di un requisito anche se di grande impatto sul sistema non va a minare per nulla la stabilità dell'infrastruttura di produzione, che è divisa in tre parti, come si vede in figura 17: *continuous exploration* (esplorazione continua dei requisiti e di eventuale nuove funzionalità), *continuous Integration* (la parte centrale del sistema, con questa tecnica il sistema è sempre in fase di "esecuzione", grazie anche ai test di integrazione) e *continuous deployment* (processo che valida le caratteristiche in output dalla Continuous integration e le esegue nell'ambiente di produzione).

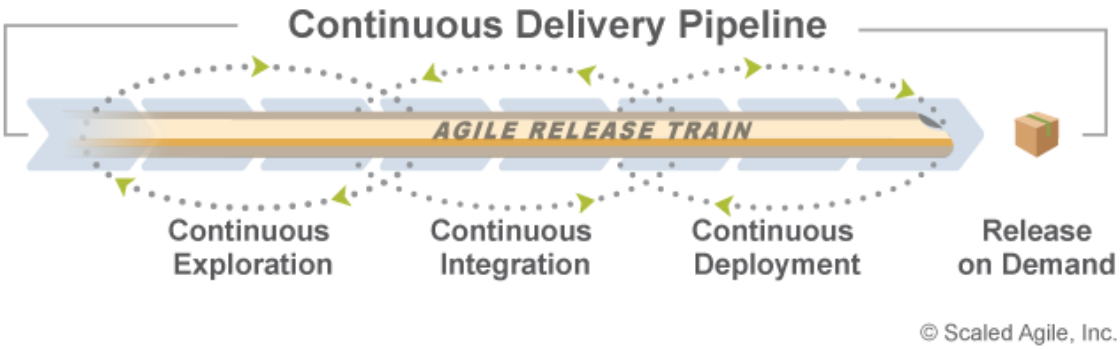

Figura 17: Le tre fasi della Pipeline di Continuos Delivery

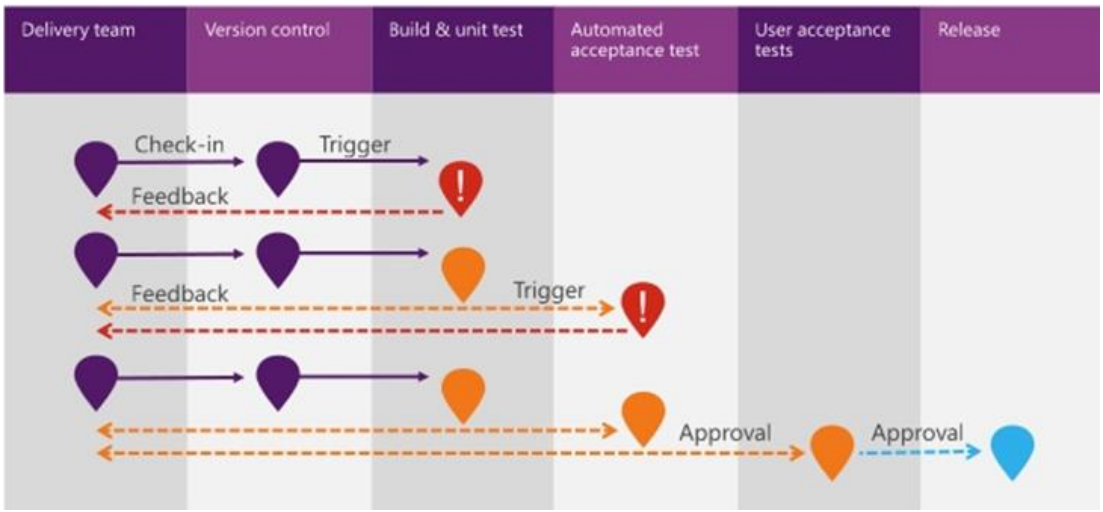

Figura 18: Il software non viene rilasciato finchè i test non sono stati approvati in tutte le fasi

2.3.3: Scrum, User Stories e Abuser Stories

Scrum [CAS, 2010] viene definito come un vero e proprio framework astratto di lavoro per lo sviluppo di software, dove è possibile utilizzare diverse tecniche e tecnologie al fine di strutturare un processo concreto che abbia come primo obiettivo quello di usufruire dei benefici Agile nella produzione di un sistema software. Nato nei primi anni 2000, esso definisce tanti elementi nel suo ecosistema di lavoro, in particolare: *artefatti, eventi e ruoli* [STE et al., 2014]. Tutto l'universo Scrum ruota attorno a questi tre elementi: all'interno di un team di lavoro sono presenti diversi ruoli, quali *product owner* (d'ora in avanti *PO*, la singola persona che rappresenta gli interessi dei clienti), *scrum master* (persona all'interno del team che controlla che il processo Scrum sia svolto correttamente) e infine i *membri* del team (persone che sono all'interno del team di sviluppo, team completamente auto-organizzanti e con completa uguaglianza tra i componenti). È importante evidenziare che lo scrum master non è superiore ad alcun membro del team, ma semplicemente un membro che è anche responsabile di gestire tutti i processi Scrum. Il ciclo di lavoro è pienamente

iterativo e incrementale, segnato da eventi fondamentali: si hanno i cosiddetti *sprint*, che rappresentano le iterazioni di un processo basato su Scrum, ovvero un periodo di sviluppo che ha un obiettivo, una durata e uno scopo preciso, che nella maggior parte dei casi è quello di andare a gestire il *product backlog*, artefatto che rappresenta la lista di funzionalità o requisiti che dovranno essere presenti all'interno del sistema. In realtà lo sprint è diviso in più parti, dove inizialmente abbiamo un evento di *sprint planning* nel quale viene deciso tra i diversi ruoli come impostare lo sprint, generando un artefatto chiamato *sprint backlog* sul quale il team poi lavorerà per una certa durata (*sprint execution*), ed infine ci saranno due eventi, *sprint review* e *retrospective*, nei quali rispettivamente si mostra quanti e in quale maniera sono stati realizzati i requisiti dettati dalla fase di backlog, e dove si discute e si esplicitano i vari problemi che sono stati riscontrati nelle fasi di lavoro, facendo delle stime e utilizzando i feedback di queste due fasi per migliorare l'intero processo di sviluppo all'iterazione successiva. Il team è completamente auto-organizzato nella produzione di vari artefatti durante il ciclo di lavoro: oltre al product backlog prodotto dal PO, vengono generati la *Board* di prodotto, che descrive la storicizzazione delle varie versioni e delle varie funzionalità del sistema prodotto, il *Burndown Chart*, diagramma cartesiano che va a rapportare la velocità media del team di sviluppo in relazione a quella sullo sprint attuale, ed infine uno degli strumenti più importanti che rappresenta la concretizzazione formale del *Product Backlog*: le cosiddette storie, o *User Stories*.

Una *User Story* è essenzialmente una breve descrizione che il cliente delinea nel momento in cui descrive le caratteristiche e le funzionalità che vorrebbe sfruttare nell'applicazione: questo processo in realtà è molto semplice e ricorrente, sebbene prima dell'utilizzo di queste storie rimaneva implicito nella mente del committente, a svantaggio di una comunicazione chiara e trasparente tra le parti con i vari aspetti negativi già descritti.
Seguire queste storie in fase iniziale del progetto garantisce senza dubbio innumerevoli benefici, tra gli altri:

- Viene subito pensato l'applicativo dal punto di vista dell'utilizzatore finale;
- Possono essere aggiunte, modificate ed eliminate storie senza problemi nel processo;
- Le storie possono catturare diverse funzionalità;
- Le storie possono essere usate per basare dei planning (di risorse, di tempistiche);

Il più grande vantaggio però è che non ci si focalizza a *COME* implementare una certa funzionalità, ma si mette da subito il punto di riferimento a *CHI* desidera questa funzionalità e al *PERCHE'*, creando un territorio comune e una base solida tra tutti gli attori aziendali che in questo modo sono in grado di comunicare sullo stesso piano.
La sintassi classica di una storia è il seguente:

- AS A <ruolo>
- I WANT <fare qualcosa>
- SO THAT <possa ottenere valore per il mio business>

Un esempio concreto potrebbe essere la situazione di un Manager che abbia bisogno di uno strumento Web che gli permetta di cercare le ore lavorative di un certo impiegato, dato il suo cognome:

- AS A <Manager>
- I WANT TO <Cercare gli impiegati dato il loro cognome>
- SO THAT < Stabilire le loro ore lavorative mensili>

Per quanto possa risultare a primo impatto molto semplice descrivere un insieme di storie di questo tipo, in realtà il giusto livello dettaglio di una storia non è un aspetto né globalmente condiviso, né banale (sull'esempio: cosa si intende con l'azione di cercare? In che modo gli impiegati hanno associate le ore lavorative mensili? Come sono calcolate queste ore?). L'ambiguità di uno schema di questo genere è una caratteristica voluta proprio per fare comunicare cliente, esperto del dominio, e tecnici, esperti delle tecnologie. Tra le varie caratteristiche che devono avere le storie, definite sotto l'acronimo *INVEST* (Indipendenti, negoziabili, di valore, stimabili, piccole e testabili), senz'altro la più importante rimane quella dell'essere testabili: ci deve essere un modo formale per validare queste storie, e avere un riscontro univoco sulla loro completezza e correttezza. Per ogni storia abbastanza complessa in Scrum in realtà è possibile suddividerla in diversi task, più tecnici (sull'esempio, i diversi task potrebbero essere: progettare un metodo che ritorna la lista degli impiegati, progettare struttura che tiene conto di tutti gli impiegati…), e infine, per verificare che sia stata effettivamente implementata e dia il risultato atteso, ad ogni storia è associata la scrittura immediata dei cosiddetti test di *acceptance criteria*, ovvero un elenco puntato di test che si va a depennare nel caso in cui essi vengano verificati formalmente (con un vero e proprio programma, metodo o strumento applicativo).

È proprio in questa fase molto delicata che si va a catturare tutto ciò che il cliente si aspetta di avere alla consegna dell'iterazione, ed è qui che ci si concentra sul *COSA* e si vanno ad esplicitare tutte le assunzioni fatte, verso l'implementazione corretta della funzionalità stessa in un'ottica pienamente Test Driven.
È nella scrittura degli *acceptance criteria* che viene eliminata qualsiasi ambiguità nello sviluppo della nuova funzionalità, nella forma:

- GIVEN < scenario specifico di utilizzo della funzionalità>
- WHEN <situazione specifica di utilizzo>
- THEN <ciò che mi aspetto dalla funzionalità>

In questa fase è possibile associare tanti test di accettazione ad una certa storia, a qualsiasi granularità e livello di dettaglio.
Sull'esempio precedente, uno dei tanti *acceptance criteria* potrebbe essere:

- GIVEN <lista impiegati>
- WHEN <inserisco il cognome di un impiegato>
- THEN <visualizzo correttamente le sue ore di lavoro>

Questo in realtà è un test sull'intera funzionalità, ma a loro volta i test possono essere suddivisi su più livelli, sull' esempio: testare la corretta memorizzazione della lista impiegati, il corretto meccanismo di input per il cognome, il corretto filtraggio sull'elenco, e via dicendo.

In questo mondo, come è affrontata la questione sicurezza?

Da sempre nella metodologia di lavoro Scrum e in letteratura ci si è posto questo problema, proprio perché riuscire a mettere i requisiti di sicurezza nella fase prototipale del software non solo costa meno di eventuali patch o aggiornamenti in una fase avanzata del software, ma proprio perché molte volte è possibile solo in questa fase comprendere a pieno le interazioni e gli aspetti comportamentali di un componente o di una funzionalità che poi inseriti in un contesto ampio saranno più complessi e difficili da definire.
Sono state proposte diverse strade: in letteratura [ZUL et al. , 2011] viene proposto ad esempio l'approccio "Every-Sprint", che prevede l'inserimento di un vero e proprio ruolo di Security Master all'interno del processo, e artefatti come il Security Backlog, che appesantiscono un po' l'intero processo dato che si va a valutare l'impatto sulle situazioni di sicurezza presenti in tutte le diverse funzionalità descritte nelle User Stories. In altre pubblicazioni [NGU, 2015] viene invece proposto anche un approccio un po' più leggero chiamato *Secure Scrum*: non si va ad integrare la sicurezza su tutte le user stories, ma viene aggiunto uno spike di sicurezza come se fosse un evento Scrum, quindi con meno impatto sul processo, ma ovviamente più rischi dato che non definiamo la sicurezza tra le funzionalità base del sistema. Per quanto dunque sia importante definire la sicurezza sulle singole storie, l'overhead introdotto da un intero nuovo ecosistema (con la figura del Secure Master) potrebbe appesantire oltremodo il processo Agile, andando a minare i benefici di questa metodologia (rilasci veloci, iterativi e continui). Esiste un terzo modo per affrontare la questione sicurezza in Scrum è il metodo che forse racchiude in sé il tradeoff più equilibrato tra le diverse proposte: *Abuser (or Evil) Stories* [PEE, 2008] [THO, 2014].
Nonostante il termine minaccioso che lo descrive, il concetto di *Abuser Story* è forse la modalità più naturale e semplice di andare a integrare sicurezza sul mondo Scrum (che, come già descritto, è palesemente orientato verso la semplicità): essa identifica qual è la maniera in cui un attaccante può andare ad "abusare" di una *User Story*, violare il sistema ed estorcere a suo piacimento le funzionalità descritte da una storia. Sono essenzialmente le storie viste da un punto di vista di un attaccante: come user stories, esse si concentrano su chi agisce (avversario), sul perché agisce (per ottenere vantaggi o benefici), su come agisca e, soprattutto, come per le storie, queste *abuser stories* sono ugualmente brevi, indipendenti, di valore, negoziabili, e soprattutto testabili.
Si riassumono in tabella i vari approcci.

APPROCCIO	Pro (V)	CONTRO(X)
Every-Sprint	Inserimento del ruolo di Security Master nel processo garantisce sicurezza su tutto il ciclo di lavoro.	Il processo Scrum diventa troppo pesante
Secure Scrum	L'inserimento della sicurezza come semplice "spike" o evento Scrum riduce l'impatto sul processo	Rischio elevato di non considerare le situazioni più importanti di sicurezza sul processo
Abuser Story	I requisiti di sicurezza vengono associati direttamente alle funzionalità che devono essere implementate: equilibrio tra overhead e sicurezza	Tutto ciò che non è definito in fase in fase prototipale non verrà mai controllato. Nuovi scenari di attacco non sono gestiti runtime

Tabella 6: Approcci proposti in letteratura per sicurezza in Scrum

Ritornando all'esempio precedente, ci potrebbero essere diversi scenari di violazione della *user story* definita, ad esempio [OWA2] :

- AS A <hacker>
- I WANT <mandare "bad data" nell'URL di ritorno alla ricerca, così che posso accedere a dati e funzioni per le quali non sono autorizzato>
- SO THAT <leggere dati sensibili e ottenere informazioni>

Questo è solo uno dei tantissimi attacchi (*Semantic Url Attack*) che potrebbero violare la storia definita precedentemente, dato che tanti altri scenari di attacco potrebbero essere descritti: in ogni caso, l'importanza di esplicitare a questo livello prototipale i possibili scenari di attacco inizia a essere di grande valore, soprattutto nel momento in cui queste storie possano essere effettivamente testate e verificate. Dualmente al concetto degli *acceptance criteria* per le user stories, potremo definire sulla stessa falsa riga diversi test che vadano a controllare la funzionalità descritta da questa storia malevola, andando a verificare se effettivamente essa sia valida nel nostro sistema oppure no: viene definito il concetto di *rejection criteria* come l'insieme di test che vanno a verificare l'effettiva validità di questa operazione malevola, e che quindi restituiscono un risultato positivo proprio quando questa storia non sarà più valida (per questo il termine *rejection*), al contrario dei classici test di accettazione.

Sull'esempio, potrebbe essere un *rejection test*:

- GIVEN <manager che chiede lista impiegati>
- WHEN <server elabora pagina di risposta>
- THEN <cambiare l'URL di risposta e mandarlo al manager>

La funzionalità descritta dalla *User Story* quindi non sarà completata solamente quando tutti gli *acceptance criteria* saranno validati, ma sarà considerata completata quando anche i diversi *rejection criteria* definiti a partire dall' *Abuser Story* associata saranno verificati. E' necessario ovviamente in questo scenario il dialogo tra gli attori aziendali, e in particolare tra sviluppatori ed analisti di sicurezza che si occuperanno dell'implementazione dell'eventuale contromisura: è fondamentale inoltre, soprattutto in ambito IoT, essere in grado di costruire una vera e propria infrastruttura per la verifica della vulnerabilità dato che, molto spesso, i test potrebbero avere a che fare con vere e proprie operazioni "basso livello", come l'eventuale verifica di manomissione del dispositivo. I limiti sono ovviamente l'appesantimento del processo tradizionale Scrum, che però produce come output test automatici sulla sicurezza che, se progettati bene, potrebbero essere riusabili ed estendibili su altre situazioni, e inoltre è da tenere in considerazione il problema di una difficile previsione su eventuali scenari malevoli che potrebbero venire fuori in una fase matura del sistema. Infatti, tutto ciò che è stato definito attraverso i test sulle *abuser stories* sarà caratteristica peculiare del software, ma tutto ciò che è stato lasciato fuori non è ovviamente incluso e prevedibile, anche se si vedrà che a questa tipologia di approccio può essere collegata una suite di

strumenti in grado di gestire questa specifica problematica (strumenti di previsione che sfruttano il Machine Learning, per esempio).

Riassumendo viene definito in figura 19 il flusso generale di lavoro, dopo che la funzionalità dell'User Story è stata completata: definizione dello scenario di attacco e requisito reale (Abuser Story), progettazione del(i) *Rejection Test(s)* sulla particolare Abuser Story, verifica dei test, gestione del risultato ed eventuale progettazione della contromisura, verifica dell'effettivo funzionamento della contromisura, ed infine rilascio della funzionalità.

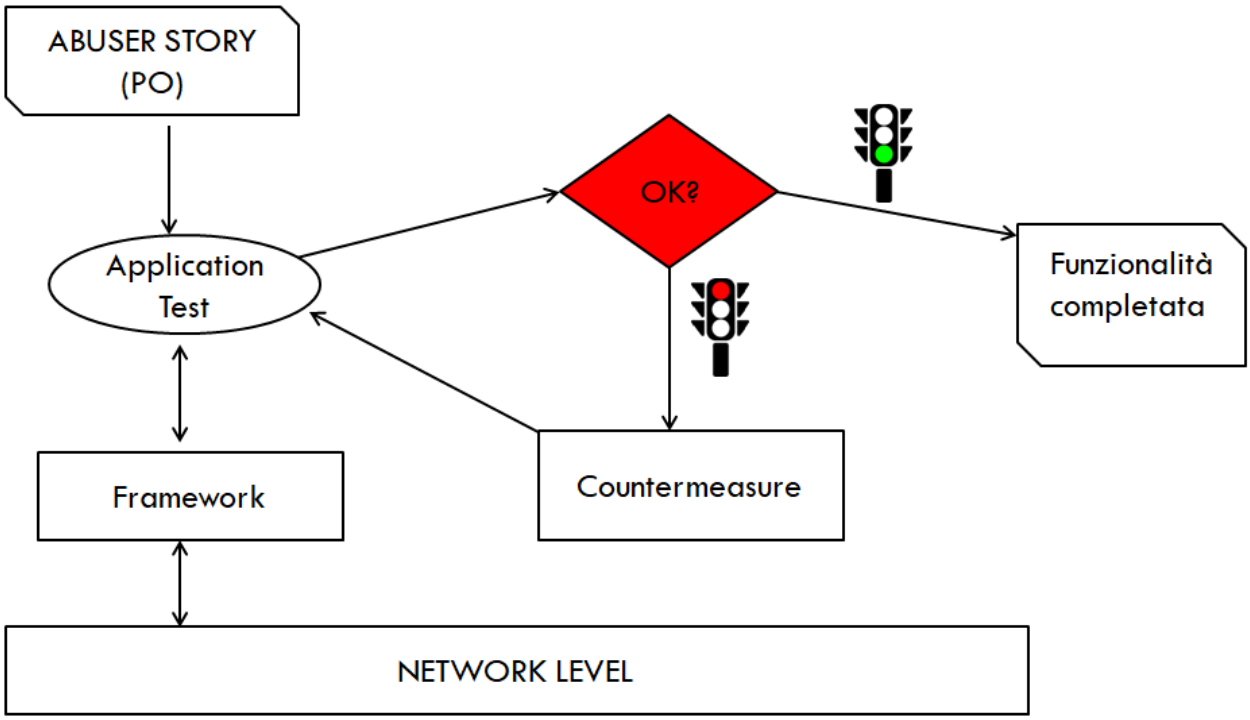

Figura 19: Flusso Abuser Stories IoT

2.3.4: Security By Design

Dopo aver descritto l'ecosistema Agile, cercando di mettere in risalto i punti fondamentali che verranno poi utilizzati per la progettazione di una soluzione reale, e in particolare dopo essersi soffermati su ciò che è più importante per un percorso relativo alla sicurezza come il concetto di Abuser Stories, si va a definire in questo paragrafo quale potrebbe essere una possibile soluzione generale di sicurezza per IoT, descrivendola ad alto livello e illustrando nel prossimo capitolo come effettivamente lo stack, la metodologia e la teoria generale proposta sia valida o meno su di un caso reale prototipale IoT aziendale. In particolare, in merito alle varie caratteristiche che un sistema software IoT deve avere (visibilità, trasparenza, sviluppo e rilascio veloci e dinamici, sicurezza), le

quali possono essere garantite dall'utilizzo dell'approccio Agile con l'inserimento di Abuser Stories, occorre delineare strategicamente uno schema risolutivo nuovo all'interno del team [KOM, 2016] che dia lo strumento giusto a un team di sviluppo software e analisti in grado di integrare pienamente il ciclo di lavoro di Abuser Stories descritto nel paragrafo precedente in un'infrastruttura concreta e consolidata: per questo motivo, facendo riferimento alla pipeline di Continuos Delivery, viene definito il concetto di **Security By Design** come l'inserimento di test automatici relativi alla sicurezza all'interno della pipeline di produzione del software. I test che verificano gli scenari dettati dal ciclo di lavoro descritto dalle Abuser Stories diventano parte integrante del processo di continuos delivery, diventando così elemento centrale nella produzione del sistema software stesso, costruendo le diverse funzionalità verificando prima del rilascio gli scenari di sicurezza che sono stati definiti attraverso questi test, nel flusso mostrato in figura 20([LIN, 2017]).

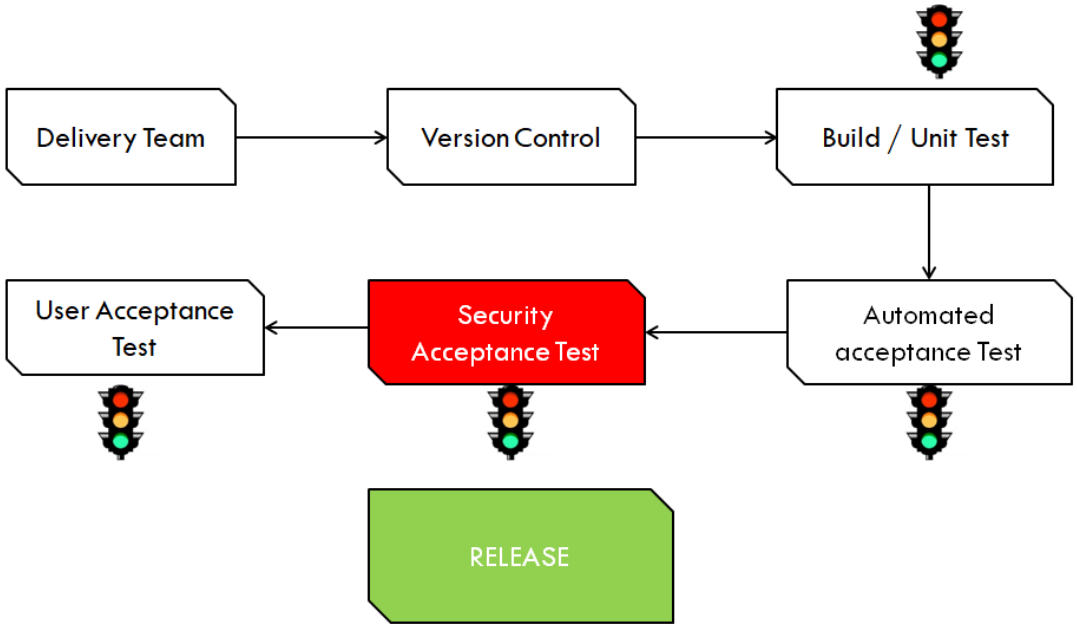

Figura 20: Pipeline di Continuos Delivery con l'inserimento dei test sulla sicurezza per lo scenario Security By Design

Con questo schema di produzione, il software che verrà rilasciato sarà per definizione "immune" agli scenari di attacco definiti come Abuser Stories, e lo sarà per qualunque modifica fatta in futuro dato che, per definizione, vengono continuamente mandati in esecuzione i test automatici che vanno a testare questi scenari, e la natura di Scrum ci garantisce comunque che questo processo non va ad appesantire alla lunga il processo di produzione del software, anche se in una prima fase di sviluppo ci deve essere assolutamente un coinvolgimento e una trasparenza di lavoro tra sviluppatori e analisi di sicurezza. L'unico limite abbastanza evidente di un approccio del genere è che non potrebbe essere utilizzato così com'è su scenari di applicazioni legacy e già molto complesse, proprio perché dipende fortemente dal fatto che il sistema software sia ancora in fase di prototipo: inoltre, anche se le abuser stories vengono scritte formalmente da un product owner che dovrebbe avere una visione completa e ordinata del prodotto in questione, ovviamente è impossibile definire tutti gli scenari di attacco a priori, come già ribadito precedentemente. Con questo schema generale, però, potremo

effettivamente utilizzare strumenti che non siano obbligatoriamente test a posteriori sulla funzionalità, ma bensì associare anche tecniche "predittive" per poter capire sulla base dei dati raccolti quanta probabilità abbiamo di avere scovato un'anomalia all'interno del sistema. È proprio su questa tematica che oggi machine learning e cyber security lavorano insieme, e attraverso dei test automatici sulla sicurezza potrebbero essere inseriti anche algoritmi puntuali che siano in grado di determinare, prevedere e magari isolare situazioni anomale che sono impossibili da determinare a priori, andando a dare un feedback immediato agli sviluppatori e soprattutto andando a isolare il componente attaccato.

Per questi motivi si è deciso di andare ad affrontare all'interno dell'azienda l'analisi, la progettazione e la realizzazione di un caso di studio reale andando a calare questo schema generale in un prototipo reale: l'intero lavoro viene descritto nel capitolo tre, e andrà a valutare se effettivamente questa soluzione proposta potrebbe essere valida su di un caso reale, con dettagliate conclusioni, risultati e spunti per il futuro.

Capitolo 3: Realizzazione di un sistema di sicurezza sul prototipo IoT Moovbit

3.1: Architettura del prototipo aziendale sullo Internet of Things

Il progetto di Tesi è stato realizzato all'interno del reparto R&D dell'azienda Scai Consulting (Gruppo Scai). In particolare il lavoro si inserisce nell'ambito della messa in sicurezza di alcuni aspetti di un prodotto IoT sperimentale, chiamato Moovbit [CAR, 2017], realizzato per il campo wearable.

MoovBit è un sistema prototipale IoT che raccoglie dati biometrici attraverso dei sensori (accelerazione, giroscopio), nato all'interno dell'azienda per integrare un prodotto aziendale preesistente chiamato F-Trainer, che è a sua volta un'applicazione Web per la gestione degli allenamenti sportivi. L'obiettivo è stato quello di popolare questo prodotto anche con dati relativi al movimento (contapassi, calorie bruciate), per rendere il servizio più completo ed estenderlo con nuove funzionalità, ad esempio con analitiche accurate di profilazione utente.

Vengono illustrati in figura i brand di prodotto.

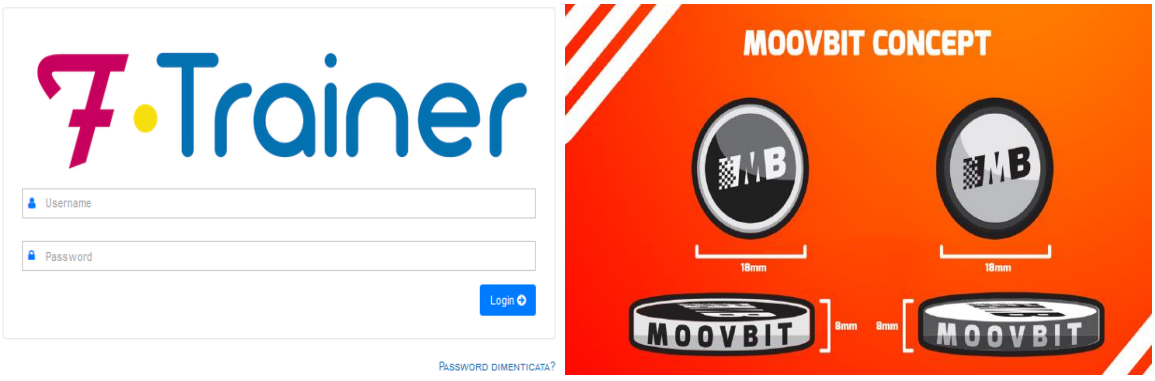

Figura 21. A sinistra: Schermata login di F-Trainer. A destra: Concept del brand Moovbit

L'intera architettura Moovbit è il frutto di un lavoro di progettazione attento verso le proprietà di scalabilità, estendibilità e performance: in particolare, esso presenta un'architettura IoT general purpose a tre livelli (figura 22), che potrebbe essere riutilizzata in molti domini applicativi.

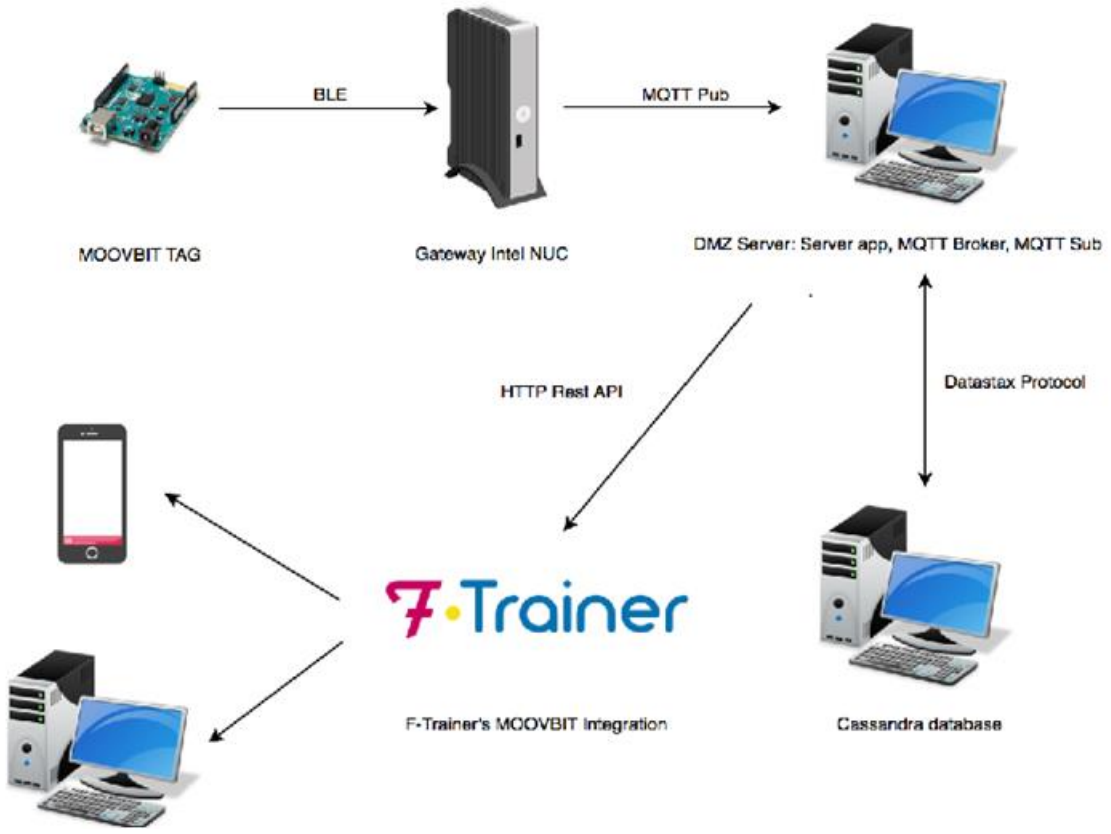

Figura 22: Integrazione architettura Moovbit con prodotto F-Trainer, a cura di Ing. Carpignoli Nicolò

Nel dettaglio, la raccolta dei dati legati al movimento (accelerometro, giroscopio) viene effettuata da un dispositivo Moovbit Tag (Arduino [ARD]) che attraverso il protocollo BLE espone queste informazioni verso un'entità di elaborazione centrale, chiamata Gateway (Gateway Intel Nuc) che a sua volta, oltre a piccoli calcoli computazionali sui dati raccolti e una prima memorizzazione attraverso un particolare registro, va a pubblicare attraverso il protocollo MQTT [MQT] i dati su di un Server aziendale interno (DMZ) che, oltre a salvare questi dati non strutturati in un DB NoSQL Cassandra [CAS], va ad integrare attraverso Rest API questi dati sull'applicativo preesistente. Senza andare nel dettaglio di tutta l'architettura, si va brevemente a spiegare quali sono i problemi più importanti per quanto riguarda la sicurezza, e perché sono state effettuate determinate scelte nella progettazione di questo sistema IoT.

Ci possono essere problematiche di sicurezza su tutta l'architettura: ad esempio, se le comunicazioni non sono messe in sicurezza (crittate) esse possono essere comprese da chiunque, favorendo la condivisone di informazioni anche molto sensibili, e inoltre non essendoci strumenti per l'identificazione ed autenticazione, chiunque può spacciarsi per un elemento fidato dell'architettura e interagire con il sistema: da considerare soprattutto l'eventuale attacco al Gateway o al Server, che se non hanno meccanismi di protezione possono dare libero accesso a un avversario capace di andare a manomettere senza problemi il dispositivo IoT (che oggi è un

prototipo wearable, e domani potrebbe essere un sensore per una pompa di insulina o qualcosa di più critico).

BLE nella prima parte dell'architettura consente un trasferimento di dati in un'area abbastanza corta, in tempo reale e a basso consumo di energia, fondamentale per limitare i costi in fase iniziale: il gateway utilizza un registro per implementare un semplicissimo meccanismo di autenticazione dei vari possibili dispositivi che può gestire, e a sua volta dopo aver elaborato i dati va a esporli attraverso MQTT ancora una volta per motivi di performance e scalabilità, data la leggerezza del protocollo usatissimo in tanti domini IoT [POZ, 2015]. Infine, il server DMZ va a memorizzare questi dati utilizzando un DB NoSQL, considerando la possibilità eterogeneità e mole di dati in ingresso al sistema [ASA, 2014], e a renderli disponibili all'applicativo web tramite Rest API HTTP, che è stato scelto sempre per motivi di performance e soprattutto scalabilità. In questa architettura prototipale la questione sicurezza è rimasta solamente teorica e non è stato implementato alcun meccanismo per garantire alcun requisito di sicurezza, a meno del semplice registro per l'autenticazione lato gateway. È quindi possibile elaborare un primissimo piano d'azione per individuare quali potrebbero essere nel dettaglio i punti sul quale definire requisiti di sicurezza all'interno dell'architettura, come descritto in tabella 7, soprattutto nella comunicazione e nella memorizzazione dei dati, per poi andare a definire effettivamente qual è lo scenario o gli scenari che l'azienda vuole proteggere e andare più sul dettaglio tecnico di lavoro.

INTERAZIONE	SCENARIO	EVENTUALE PROTEZIONE
Device -> Gateway	Comunicazioni non crittate; Protocollo di autenticazione non presente lato Device	Comunicazioni/ Autenticazione: Algoritmi embedded [REE, 2017]
Gateway -> Device	Comunicazioni non crittate; Protocollo di autenticazione presente lato Gateway (ma debole)	Comunicazioni : Protocollo crittografico (RSA es.); Autenticazione: potenziamento protocollo autenticazione con registro (username e password)
Gateway -> Server	Comunicazioni non crittate; Autenticazione non presente	Comunicazioni e Autenticazione: TLS per MQTT
Server -> Gateway	Comunicazioni non crittate; Autenticazione non presente	Comunicazioni e Autenticazione: TLS per MQTT
Device	Manomissione possibile	Protocollo di autenticazione
Gateway	Memorizzazione dati insicura	Protocolli (livello rete) per memorizzazione
Server	Memorizzazione dati insicura	Protocolli (livello rete) per memorizzazione; Possibilità di configurare sicurezza in DB Cassandra

Tabella 7: Possibili scenari critici nell'architettura Moovbit

In questa primissima e breve analisi si evince come è possibile inserire sicurezza su diversi livelli architetturali: in particolare, per tutto ciò che riguarda la seconda parte dell'interazione architetturale(Gateway / Server) e la memorizzazione dei dati su queste entità, troviamo tante strade che possono essere intraprese, come l'utilizzo di TLS e SSL in MQTT per la confidenzialità, integrazione e autenticazione dei dati [HMQ, 2006] e protocolli a livello di rete per la memorizzazione, in particolar modo anche su diversi livelli configurabili in Cassandra, facendo pienamente affidamento alle capacità computazionali delle due entità. Capacità computazionale che invece risulta limitatissima nella prima interazione, tra dispositivo e gateway. Se lato gateway è possibile potenziare il registro con modalità di autenticazione utilizzando per esempio credenziali come utente e password e controllando possibili violazioni del meccanismo (es. rilevazione forza bruta), lato dispositivo abbiamo basse capacità computazionali per gestire protocolli classici di questo tipo: per questo motivo molte volte si propone un'opzione "embedded", ovvero fornire i dispositivi con già dentro meccanismi per garantire criteri di sicurezza. Il problema, come già descritto, è che molti di questi dispositivi sul mercato non prevedono nella maggior parte dei casi alcun meccanismo: per questo motivo, andare ad affrontare un percorso completo sull'analisi della prima interazione risulta essere più interessante per l'azienda, soprattutto per quanto riguarda la tematica autenticazione e manomissione. Oggi il dispositivo è utilizzato per un dominio wearable, e un'eventuale manomissione porterebbe come conseguenza più grave l'inutilizzo di uno smartwatch: eppure se un domani l'azienda volesse ampliare i suoi mercati di riferimento IoT con lo stesso effort tecnologico(es. smart home, smart hospital o smart car), una manomissione potrebbe avere conseguenze molto più critiche, e quindi è giusto affrontarla come vero e proprio requisito di riferimento.

3.2: Analisi dei requisiti e gestione flusso di lavoro

Vengono definiti quindi quali sono i requisiti di sicurezza e come impostare la gestione del flusso di lavoro sul prototipo Moovbit, in relazione alla metodologia di sviluppo software Agile Scrum con la quale è stato costruito. Ci si vuole concentrare sulla prima parte dell'interazione architetturale, in particolare sulla comunicazione tra dispositivo IoT e gateway, andando a voler garantire requisiti di **autenticazione e di non manomissione** del dispositivo (*T: Tampering*). A questo punto, facendo riferimento al lavoro di progettazione descritto nel capitolo precedente, si vuole andare a definire un flusso di lavoro che sia in grado non solo di andare a garantire questi requisiti all'interno del prototipo IoT, ma che sia in grado di costruire un infrastruttura in linea con la metodologia delle *Abuser Stories* in maniera tale da creare una serie di test e operazioni che andranno in fasi successive ad essere integrate con il lavoro di Continuous Delivery preesistente del software, testando nel futuro in autonomia questi requisiti. In questo scenario, occorre definire un flusso di lavoro che dapprima va a verificare attraverso specifici test le vulnerabilità descritte dalle storie che

vanno a implementare i requisiti di sicurezza voluti, controlla l'effettivo esito, e in base al risultato va ad implementare eventualmente una particolare contromisura, che dovrà essere validata: solamente dopo questa validazione la funzionalità del sistema verrà ritenuta completata. Rimanendo in ambito IoT si ha a che fare con protocolli di livello fisico e di rete per lo scambio dei dati (esempio Bluetooth), ed è dunque facile prevedere che la verifica di queste vulnerabilità potrebbe passare per "bassi livelli": occorre quindi considerare che ci sia un'interazione anche con questi livelli di astrazione nella costruzione di test applicativi. In particolare, molte volte si passa per un framework o altre soluzioni "open" che vanno a gestire al posto dello sviluppatore l'interazione con il livello fisico o di rete.

Definendo i requisiti di autenticazione e di non manomissione del dispositivo IoT da proteggere con la modalità abuser stories, possiamo effettivamente andare verso un concetto concreto di *security by design*, ovvero di inserimento dell'analisi di sicurezza a partire dalla fase prototipale del software e inserimento di essa come effettivo momento all'interno della pipeline di produzione del software, ponendola come condizione al rilascio della funzionalità, a prescindere dall'ambito a cui essa faccia riferimento: si veda figura 23 per il flusso di lavoro utilizzato. Si vedrà inoltre che questa soluzione generale può avere buoni spunti concreti per scenari di manutenibilità della soluzione, tramite anche tecniche predittive di Machine Learning.

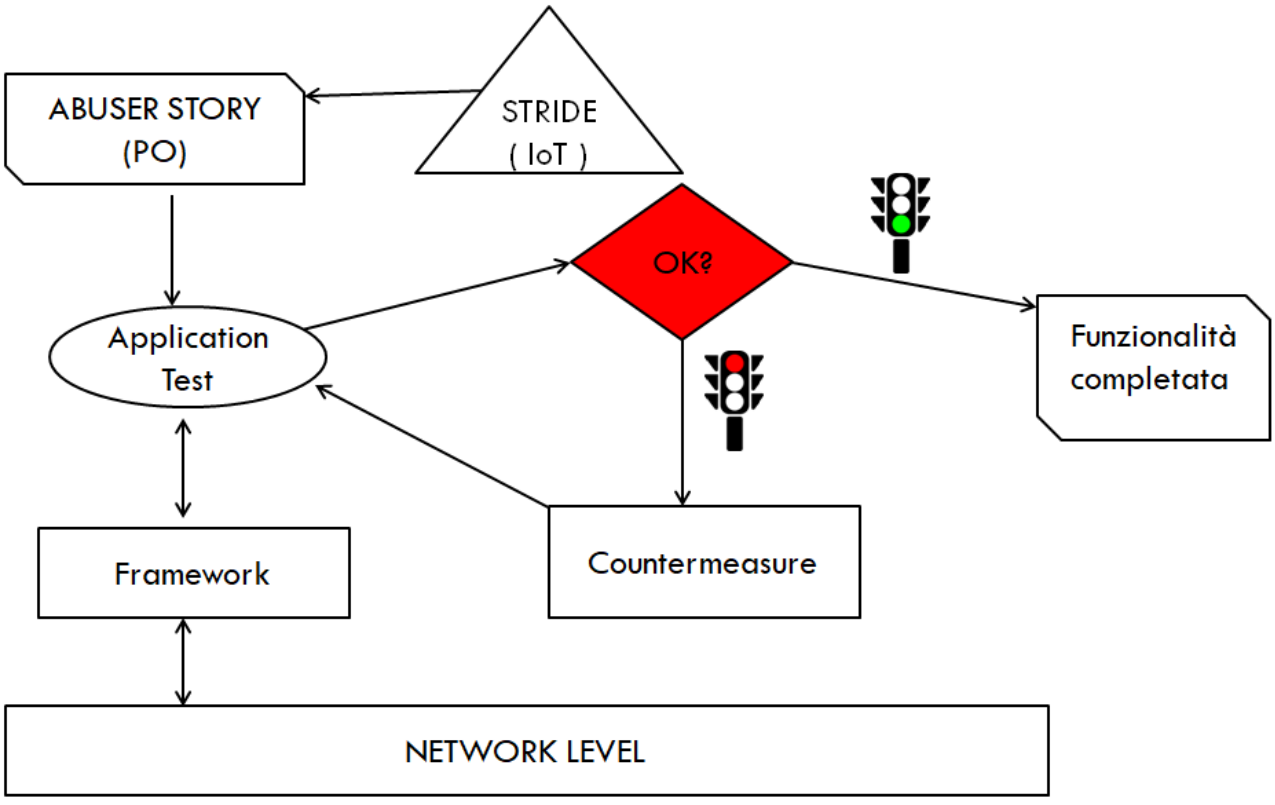

Figura 23: Esempio di Flusso di lavoro Abuser Stories progettato sul prototipo Moovbit. (Scenario: Tampering, requisito di riferimento sicurezza mondo IoT)

3.3: Security by design e Progettazione Test

In questo scenario, l'effettiva progettazione del concetto di Security by Design passa essenzialmente dalla progettazione di Test associati a delle storie, che derivano dalla fase di analisi dei requisiti descritta sopra. In particolare, nel momento in cui l'azienda vuole gestire il requisito di **autenticazione e di non manomissione** del dispositivo IoT, è molto importante andare a definire in quale momento della progettazione del software potrebbe esserci una violazione di questo requisito. Tra tutte le diverse user stories che riguardano la progettazione Scrum delle funzionalità, in particolare, si potrebbe definire un (possibile) scenario di riferimento per l'inserimento di questo requisito di sicurezza:

SCENARIO: AUTENTICAZIONE E NON MANOMISSIONE

USER STORY:

- **AS A :** Product Owner (PO)
- **I WANT :** Cercare un protocollo a basso costo di energia
- **SO THAT:** Possa scambiare dati tra il dispositivo IoT e un'altra entità (es. Gateway)

Questa storia fa riferimento alla possibilità di far comunicare i due elementi dell'architettura, ovvero Arduino e Gateway. Essa è già stata implementata e pienamente testata nel prototipo aziendale: si va a definire uno degli aspetti su cui un avversario può agire sfruttando questa storia:

ABUSER STORY:

- **AS A** avversario
- **I WANT :** Intercettare le comunicazioni
- **SO THAT:** Attaccare l'interazione con un attacco passivo, o eventualmente condurre un attacco MitM per manomettere il dispositivo

Non sapendo niente oltre al fatto che esisterà una funzionalità in grado di garantire un meccanismo di scambio di dati tra dispositivo e gateway, l'avversario può effettivamente provare a raccogliere informazioni sensibili sulla comunicazione attaccando direttamente questa interazione, con un attacco passivo o di MitM (uomo nel mezzo), mostrando l'effettiva capacità da parte sua di mettersi in mezzo alle comunicazioni senza alcuna notifica per i legittimi dispositivi, con la possibilità di raccogliere, modificare, inviare e manomettere informazioni sensibili, spacciandosi per entità fidata. In questa storia malevola è stato messo in risalto effettivamente il requisito di autenticazione, poiché la non effettiva progettazione di un protocollo di autenticazione da parte del dispositivo IoT rende questi attacchi facilmente eseguibili. La verifica di questa abuser story, e quindi l'effettiva verifica del requisito, può essere provata formalmente:

TEST:

- **GIVEN:** Dispositivo IoT
- **WHEN:** Il dispositivo è in grado di comunicare dati verso l'esterno
- **THEN :** Il dispositivo può scambiare dati solo con il corrispondente autorizzato

A questo punto, un esempio di flusso concreto per il problema specifico è stato progettato: partendo dalla funzionalità richiesta dal Product Owner, è stata formalizzata una delle possibili abuser stories relative a un requisito di sicurezza da garantire, e sono stati formalizzati le operazioni da effettuare con le quali andare a verificare se questa abuser stories è verificata oppure no. Ora è necessario andare a decidere, considerando i diversi criteri esplicitati nel capitolo precedente riguardo la progettazione, qual è la soluzione più adatta per gestire questo particolare scenario.

3.4: Interazione con il dispositivo IoT

In questo paragrafo verrà descritta accuratamente l'intera realizzazione del progetto di sicurezza per quanto riguarda lo scenario di manomissione e autenticazione del dispositivo IoT (Moovbit Tag). In particolare, verrà prima scelta la modalità con la quale andare a verificare se effettivamente questo scenario è garantito dall'interazione oppure no, andando poi in una seconda fase ad implementare una contromisura ad hoc che garantisca questo requisito, ed infine validando questa contromisura e cercando di andare a delineare come inserire questo intero flusso di lavoro in un contesto dinamico, definendo i possibili sviluppi futuri per una manutenibilità della soluzione e le attività nel breve termine da svolgere per rendere questa soluzione effettivamente un prodotto. Riassumendo, tutto parte dal voler garantire che il dispositivo IoT scambi dati e interagisca solo con/da dispositivi "fidati", e non con/da potenziali avversari, che oggi potrebbero manomettere uno smartwatch, ma che un domani con le stesse operazioni e soprattutto con la stessa complessità tecnologica potrebbero mandare in tilt ospedali, case, dispositivi medici e persino intere città o nazioni.

3.4.1: Ricerca e verifica vulnerabilità

Come primo passo del flusso di lavoro per l'utilizzo della metodologia *security by design* occorre andare a realizzare la verifica dello scenario di sicurezza, e dunque la realizzazione effettiva dei test definiti a partire dall'abuser story. Come descritto nel capitolo precedente, ci sono diversi criteri con i quali poter andare effettivamente ad implementare questi scenari applicativi in un'interazione BLE. Viene scelto di non utilizzare un analizzatore di rete per la cattura di pacchetti BLE a causa dei limiti fisici legati alla natura della tecnologia che grazie al frequency hopping[FHS] riduce la capacità di sniffing dei pacchetti, e tra le diverse soluzioni framework presenti open-source viene

deciso di utilizzare il framework Gattacker, grazie alla sua documentazione, alla sua relativa facilità di utilizzo, alla possibilità di estendere il codice con logica applicativa ad-hoc, e soprattutto grazie alla sua piena stabilità.

Dopo aver lanciato il framework su due macchine fisiche Linux dotati di due schede di rete bluetooth differenti (Central Device, Peripheral Device), e dopo aver configurato il framework in maniera tale che il Peripheral Device abbia dentro di sé tutte le informazioni del dispositivo IoT clonato (Arduino), andiamo effettivamente a realizzare degli scenari di interazione tra dispositivo centrale e Arduino, dove vogliamo leggere la caratteristica legata al dato sull'**accelerazione**, e attraverso il framework si vuole verificare la possibilità di capire se questa operazione di lettura può essere intercettata da un terzo elemento (e, nel caso positivo, se questo dato può essere modificato "on-the-fly"). Quello che succede, come è facile immaginare, è che entrambi gli scenari sono effettivamente verificati con delle prove di laboratorio:

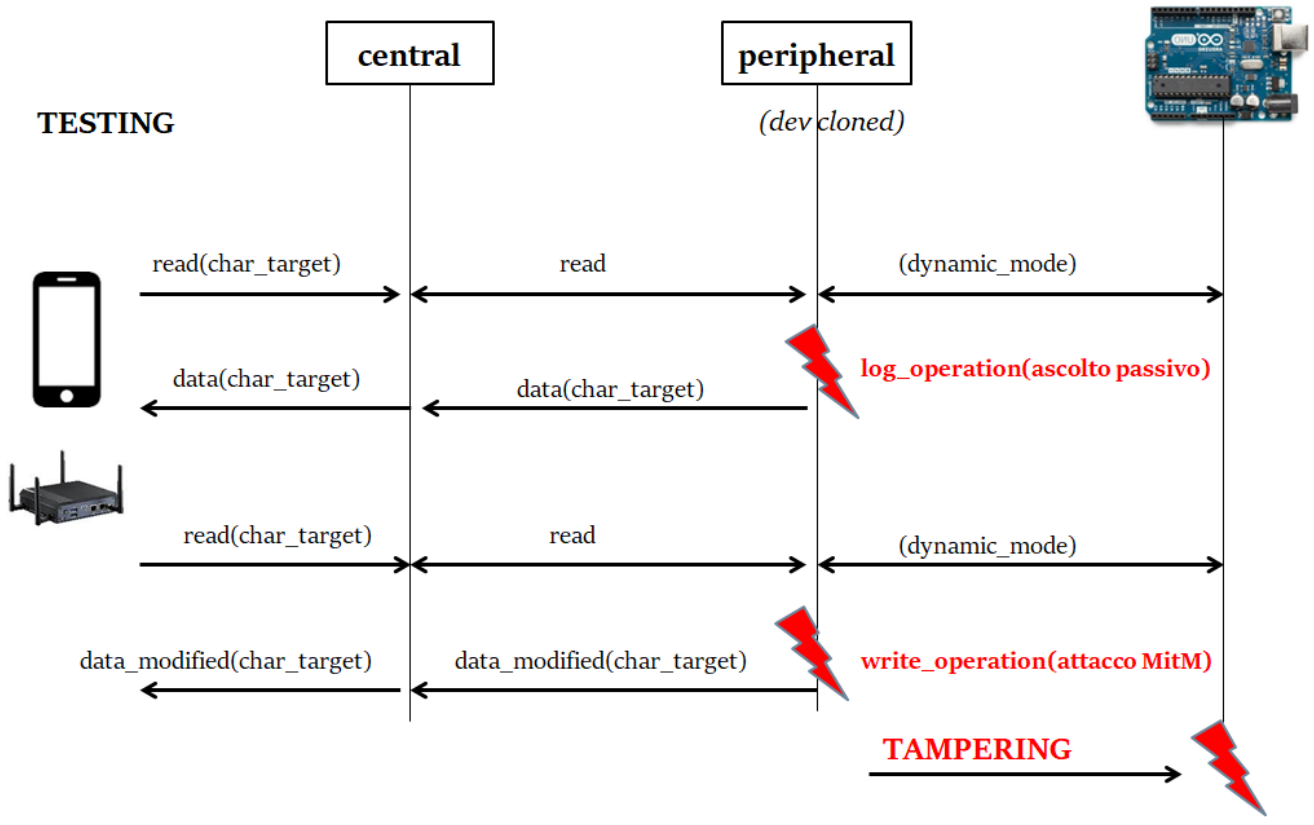

Figura 24: Operazioni di Testing per ricerca vulnerabilità con l'uso del framework Gattacker

In particolare, con riferimento a figura 24, le prove condotte sono state effettuate utilizzando 2 dispositivi centrali: il classico Gateway e un'applicazione apposita (NRFConnect [NRF]) che è presente in smartphone android, ed è stato provato che la lettura di una caratteristica target (come

può essere quella dell'accelerazione) può essere intercettata e letta senza alcun problema dal peripheral device del framework (log_operation). Inoltre, tenendo vive le comunicazioni fisiche con il dispositivo (dynamic_mode), il framework stesso può anche andare a scrivere dei dati (write_operation) ed eventualmente forzare la manomissione dell'arduino stesso (tampering) senza controllo alcuno.

Tutto il meccanismo è implementato all'interno del componente Peripheral del framework, e viene mostrato in figura 25. In questo elemento vengono registrate in fase di setup le caratteristiche del dispositivo IoT, sottoforma di file json. Su ogni caratteristica (dato) del file json è possibile definire particolari proprietà specifiche, chiamate *hooks*; ogni volta che viene letta (o scritta) la caratteristica su cui sono stati definiti degli hooks, il file json invoca la funzione JavaScript associata, eseguendola. La possibilità di sviluppare quindi logica applicativa a questo livello di astrazione garantisce allo sviluppatore (o all'analista) di andare effettivamente a concretizzare la logica applicativa di ciò che si vuole testare ogni volta che il framework lavora sulla comunicazione. Automatizzando nel futuro la fase macchinosa di setup e configurazione sia del framework che di questi file, si potrebbe arrivare a un livello in cui lo sviluppatore si concentra prevalentemente sullo sviluppare le operazioni da provare, che sono la concretizzazione delle abuser stories definite. Le librerie JavaScript con cui è scritto il framework garantiscono un buono strumento di sviluppo: si rimanda all'appendice per dettagli sul codice delle operazioni testate.

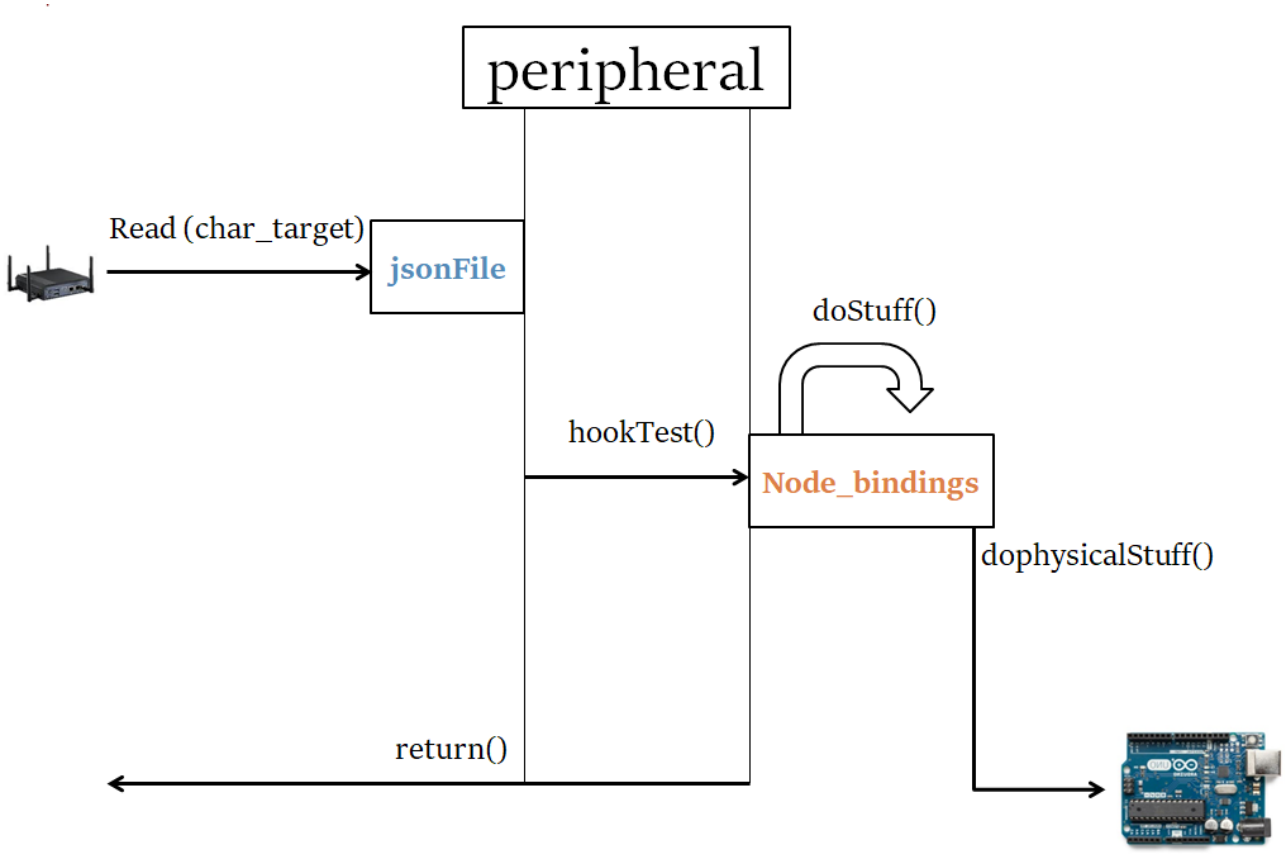

Figura 25: Dettaglio tecnologico architetturale Peripheral Device

3.4.2: Analisi tecnologica e progettazione di una contromisura per l'autenticazione

Dopo aver verificato formalmente l'esito positivo della verifica nel condurre attacchi passivi e MitM nell'interazione BLE tra i due dispositivi (con logiche possibili gravi conseguenze), e dopo aver automatizzato questo controllo in maniera tale che sia lanciato come un vero e proprio test automatico, ci si accinge (da progettisti e analisti di sicurezza) a sviluppare una contromisura che vada a mettere in sicurezza il requisito voluto, per poi andare a validarla in un processo che poi in futuro dovrà essere anch'esso automatizzato (nell'ottica di una possibile interazione con la pipeline di continuous delivery). Si discuterà poi accuratamente se sarà significativo validare la contromisura attraverso gli stessi test sviluppati su Gattacker, oppure occorrerà percorrere altre strade.

Come già espresso, uno dei problemi principali nell'andare a inserire requisiti di sicurezza nell'interazione BLE risiede proprio nella difficile programmabilità dei livelli di sicurezza all'interno del protocollo, e soprattutto nelle vulnerabilità stessa di SSP (Secure Simple Pairing) che rappresenta l'unico strumento in BLE per definire diversi livelli di sicurezza. Per questo motivo si decide all'interno dell'azienda di andare a definire una soluzione proprietaria che permetta di definire a livello applicativo ciò che si vuole ottenere, ovvero la garanzia del requisito di autenticazione lato dispositivo IoT, onde evitare manomissioni e intercettazioni: questa soluzione viene chiamata per comodità *HSEC* (HandShake Security), in quanto il nucleo di questa soluzione dovrebbe fare riferimento a un accordo applicativo (per questo, handshake), che vada a garantire nell'interazione il requisito di sicurezza voluto (autenticazione del Gateway presso Arduino). Definendo una contromisura a livello applicativo è possibile andare a risolvere due problemi: il primo che riguarda come già descritto il limite della modalità che viene offerta da BLE per la sicurezza, ma soprattutto viene risolto il problema di andare a implementare costosi e complessi meccanismi di *cross-layering,* ovvero di andare a gestire lato sviluppo direttamente il protocollo fisico BLE, nonostante i tanti vantaggi di questa tecnica in contesti WLan [KLI et al., 2006]. Viene invece delineato a monte il giusto funzionamento attraverso una soluzione applicativa: il limite visibile è rappresentato ovviamente da un gap tecnologico che occorre affrontare attraverso un'analisi tecnologica approfondita ed accurata. Inoltre, occorre stabilire un ordine, un meccanismo preciso di iterazione che potrebbe aggiungere degli overhead in termini di costo e computazione.

L'analisi tecnologica per la successiva definizione del protocollo di riferimento presentato in *HSEC* passa per due punti fondamentali: l'interazione con il dispositivo, e la soluzione tecnologica che ci permette di implementare un particolare protocollo limitando costi e problemi di efficienza computazionale. Per quanto riguarda il primo aspetto, l'unico modo per poter interagire con il dispositivo IoT è attraverso operazioni sui dati che espone tramite bluetooth, dati che come già descritto sono strutturati in maniera gerarchica in servizi e caratteristiche: nella definizione di un'interazione con questo dispositivo, l'unica modalità di passaggio di dati e informazioni avviene dunque solamente attraverso la scrittura / lettura di una di queste caratteristiche. L'interazione tra gateway e dispositivo IoT utilizza lo stack Bluez di Linux [BLU, 2000] e in particolare uno strumento linux chiamato *gattoool* [IWA] che è un tool a linea di comando in grado di manipolare i dati di un dispositivo BLE, come realizzare letture e scritture: il gateway, nel momento in cui

interagisce con il dispositivo, utilizza questo tool per andare a leggere, scrivere e per poter recuperare dati e informazioni sui dati del dispositivo IoT. Ancora più nel dettaglio, nel prototipo Moovbit viene realizzato un binding Java della libreria *gatttool* chiamato *JGatttool* [GIT5, 2017] con la consapevolezza che, volendo implementare la logica del gateway in maniera riusabile ed estendibile, occorre passare per un modello consistente e completo, e da qui la scelta del linguaggio Java per la logica lato Gateway anziché il mero uso di gatttool come linea di comando Unix. Il gap tecnologico dell'interazione viene così colmato: riferimento in figura 26.

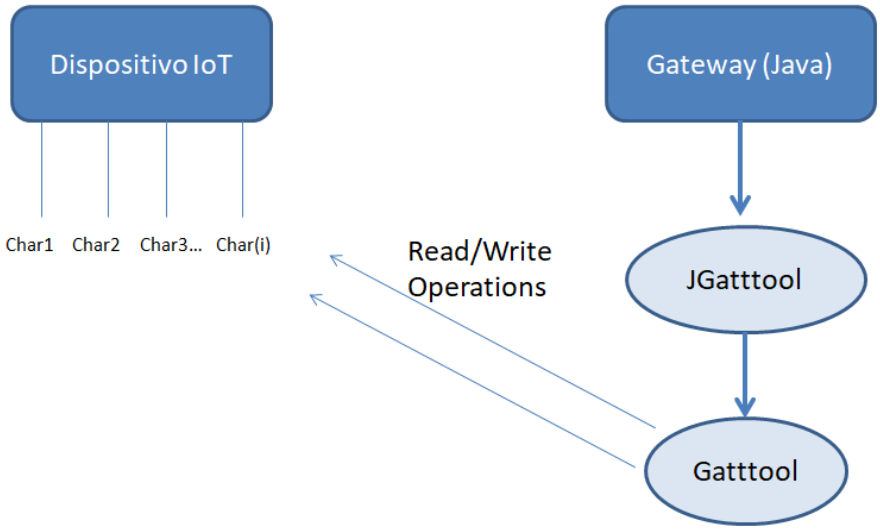

Figura 26: Realizzazione interazione tra Gateway e Dispositivo IoT

Analizzata l'interazione con il dispositivo IoT, e compreso che l'unico modo per interagire con esso per effettuare un qualsivoglia preventivo scambio di informazioni consiste nella scrittura e lettura di caratteristiche, occorre soffermarsi sulle modalità in cui il gateway possa autenticarsi con il dispositivo, e come il dispositivo possa essere dunque sicuro con chi sta parlando. L'autenticazione è uno dei principali requisiti del mondo della sicurezza informatica, e viaggia insieme a un altro requisito sulla quale da sempre sistemi software vengono costruiti, ovvero il concetto di autorizzazione (quali sono le operazioni che certi utenti sono autorizzati a svolgere in un applicativo), costruendo attraverso queste due proprietà il cosiddetto *trust relationship* all'interno del sistema software. Da sempre, l'autenticazione viene affrontata con diversi approcci: in particolare, i più utilizzati meccanismi di autenticazione vengono implementati attraverso sistemi di login (username e password) che possono essere associati anche a differenti livelli (autenticazione a due, tre fattori). Oltre alla problematica dell'eventuale vulnerabilità di un meccanismo di password non sicura, attaccabile con un semplice attacco di forza bruta, è importante notare che nell'IoT, con l'inserimento di un numero spropositato di dispositivi eterogenei, ognuno dei quali potrebbe costituire un access point per un avversario malevolo, diventa problematico andare a gestire schemi di password e accessi (mole di dati, possibile esposizione a vulnerabilità di memorizzazione): potrebbe essere quindi un'alternativa utilizzare uno schema di autenticazione che va maggiormente verso una autenticazione *machine-side*, e non più *user-side* (login user e password). Per questo

motivo, esistono schemi più orientati verso i dispositivi, che poi sono anche ripresi in soluzioni ad alto livello e "complete" come SSL e TLS (e quindi in tutto il mondo HTTPS), come ad esempio lo schema di autenticazione MAC, la cui unica implementazione attuale (HMAC) è presente in AWS (Amazon Web Services) e fornisce tanti vantaggi [WOL, 2012].

MAC fa riferimento a un protocollo di autenticazione e integrità di messaggi, e rappresenta uno dei protocolli base della sicurezza informatica: esso garantisce autenticazione e integrità di un particolare messaggio mandato da un'entità, e quindi garantisce la verifica che l'entità sia fidata.

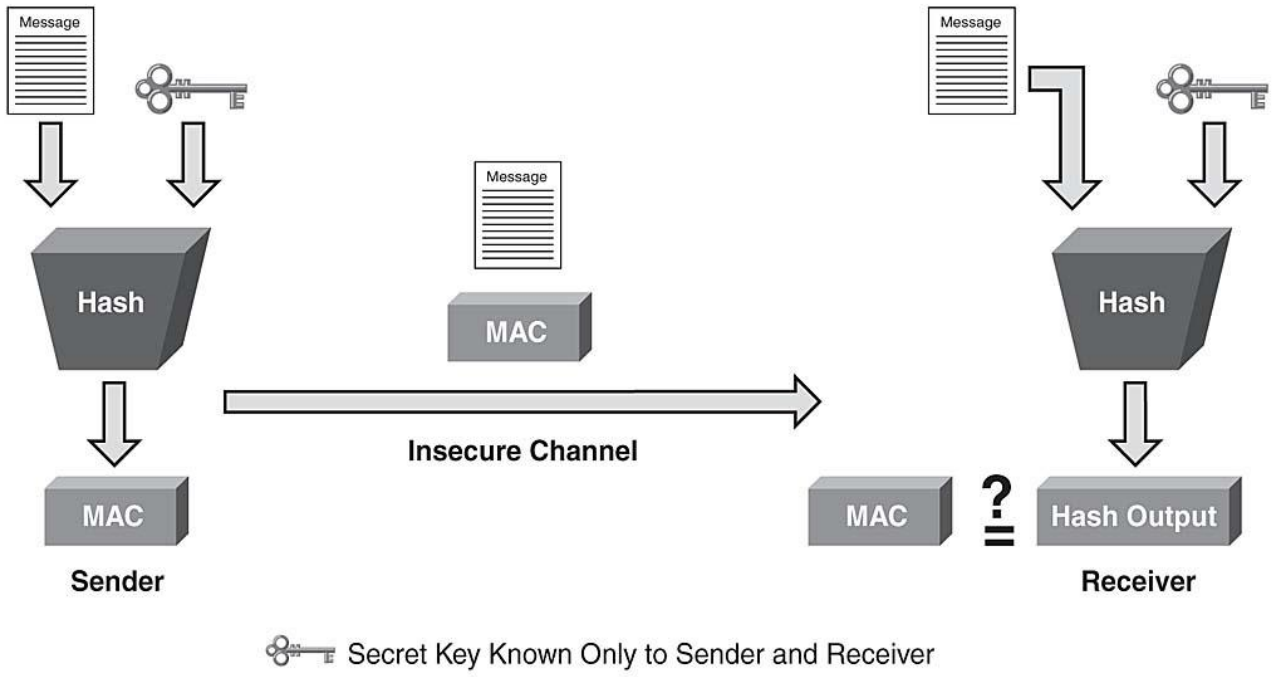

Figura 27: Protocollo MAC (chiedi a prof se Mac O HMac)

Si illustra in figura 27 il dettaglio: sia il trasmettitore che il ricevitore condividono un dato, chiamato chiave, e una funzione chiamata *hash*, con la quale viene calcolata ambo i lati sul messaggio che viaggia in rete una particolare impronta o *digest* (MAC e Hash Output). Una volta che si è sicuri che chiave e funzione hash rimangono segreti e non vengono intercettati da avversari, alla ricezione di un particolare messaggio dalla rete che pretende di essere l'impronta di autenticazione, esso viene verificato con l'impronta in locale, e in caso di equivalenza si è sicuri di parlare con un'entità fidata che condivide le nostre stesse informazioni. Questo progetto teorico può senza dubbio essere calato anche nel caso aziendale in questione: in particolare, per ottenere il requisito di autenticazione lato dispositivo IoT, il canale insicuro di comunicazione potrebbe essere l'interazione BLE, il trasmettitore sarà il Gateway e il ricevitore che controlla l'impronta sarà l'Arduino: occorre però gestire dei problemi importanti. Il primo problema riguarda il passaggio della chiave iniziale: in particolare, se i dati vengono trasmessi per la stessa interazione BLE di comunicazione, essendo questo un canale insicuro, essi potrebbero essere intercettati e rivelati

quindi a un potenziale avversario, che ottenendo queste informazioni segrete può violare tranquillamente il protocollo. Inoltre, non avendo Arduino la possibilità computazionale di sostenere un ulteriore protocollo di comunicazione (es. NFC), è impossibile passare per un altro canale che non sia BLE. Si decide dunque di pre-cablare questi dati in una fase di configurazione *wired* tra dispositivo e le varie entità che vogliono interagire con esso: così facendo, occorre anche fare l'assunzione che questa fase sia completamente sicura e che inoltre stabilisca una regola in grado di cambiare con regolarità alcuni bit di chiave (il riuso della stessa chiave può portare ad attacchi sulle frequenze). Il secondo problema riguarda la difficoltà computazionale: la funzione *hash*, infatti, fa parte della famiglia delle funzioni unidirezionali [WIK] che sono usate tantissimo in quasi ogni ambito di sicurezza, e che basano il loro comportamento sulla semplicità di utilizzare la funzione in un verso (ricavare l' impronta dalla chiave e dal messaggio) ma nello stesso tempo l'impraticabilità nel realizzare l'operazione inversa (dall'impronta, ricavare chiave e messaggio), per non permettere all'avversario di indovinare facilmente i dati segreti intercettando semplicemente l'impronta che viaggia in rete. Nonostante la relativa semplicità nell'utilizzo della funzione nel verso "giusto", il calcolo dell'hash risulta comunque uno sforzo computazionale molto importante e sono presenti (soprattutto nel mondo dei dispositivi IoT) pochissimi esempi reali che permettano di sfruttare i benefici di MAC in un dispositivo che ha basse capacità computazionali. Inoltre, è un grosso problema anche la lunghezza dell'impronta e soprattutto delle chiavi e dei messaggi che vengono gestite per il lavoro computazionale. Si va quindi a descrivere un progetto open-source che riguarda uno strumento quasi unico in questo contesto, una soluzione cross-platform chiamata *SipHash*, che verrà descritta per effettivamente comprendere le principali caratteristiche, e valutare il suo uso nel caso di studio specifico.

3.4.2.1: SipHash

Siphash [AUM et al. , 2012] è una soluzione tecnologica di funzioni che si occupano di autenticare pacchetti in rete, ottimizzata per input corti. Garantisce uno schema di autenticazione stile MAC ma più semplice, più veloce su input corti. In particolare, questa soluzione fu progettata e successivamente realizzata in diversi linguaggi di programmazione, per cercare di ottenere performance maggiori rispetto al tradizionale MAC: per esempio, su input di 16 byte (processato con un processore AMD FX-8150) riesce a ricavare una chiave in 140 cicli, ottenendo una performance molto migliore rispetto all'autenticazione MAC. L'importanza di avere a disposizione una soluzione che riesce a performare su input corti potrebbe risultare una concreta possibilità concreta per il trend IoT, popolato da dispositivi a bassa capacità computazionale: ad esempio, il dispositivo IoT a disposizione (Arduino) presenta caratteristiche(dati) BLE di massimo 16 byte. Tenendo presente questi numeri, è essenziale poter usufruire di una soluzione che ottimizza input corti. Più sul dettaglio, SipHash propone diverse fasi nella costruzione delle impronte:

- *Inizializzazione:* Vengono inizializzati 4 stati di 64 bit;
- *Compressione:* Operazione in cui vengono compressi questi stati;
- *Finalizzazione:* Padding di alcuni byte sugli stati compressi, con diversi round chiamati "SipRound";

Per quanto la modalità con cui SipHash va nel dettaglio a gestire queste operazioni sia in realtà molto complicata, il punto chiave del progetto consiste nell'andare a proporre diverse caratteristiche che superano i limiti evidenti dell'autenticazione MAC così com'è. In particolare:

- Sicurezza più elevata: Utilizzando funzioni pseudocasuali (PRF), viene assicurata l'integrità del messaggio;
- Velocità più elevata: Molto più veloce rispetto alla soluzione MAC classica;
- Semplicità, Autonomia, Minimo Overhead: I messaggi autenticati sono solamente 8 byte più lunghi dei messaggi originali.

Rispetto ad altre soluzioni che vanno a fornire una possibile implementazione del protocollo MAC, come ad esempio l'algoritmo MDA-5 presente in SSL, SipHash garantisce ottime performance per numero di byte input/cicli di processamento, risultando competitiva addirittura con classici funzioni di Hash non crittografiche (ovvero, che non si occupano di realizzare un complesso schema MAC come fa SipHash): si veda figura 28. Per questo motivo, nella progettazione di una contromisura per lo scenario di interazione BLE viene scelto di tenere in considerazione l'apporto possibile che può essere garantito da questa soluzione, facendo riferimento in particolare alla sua implementazione per Arduino e per il linguaggio Java [FOR, 2013].

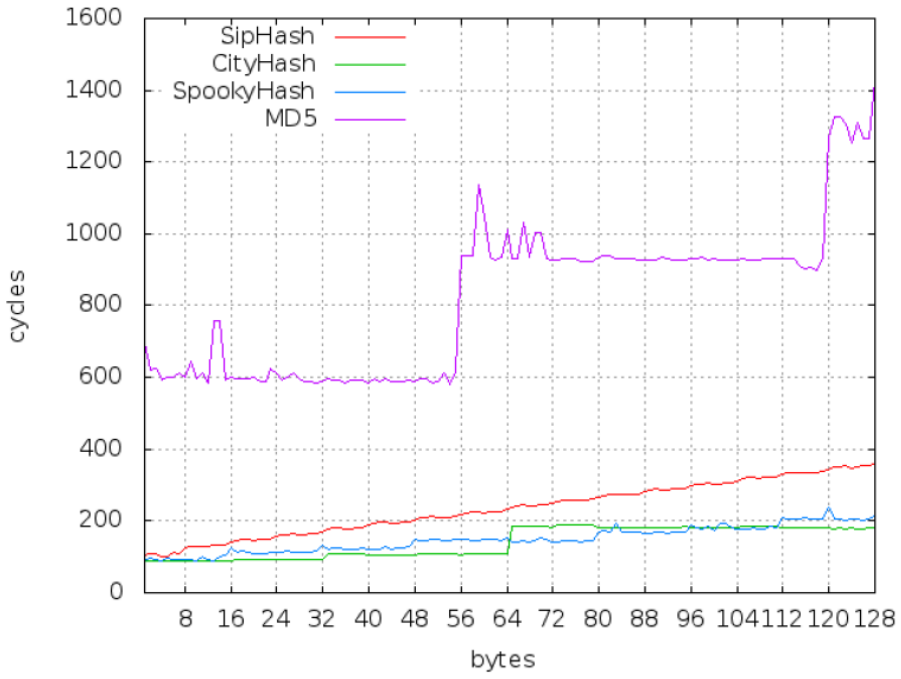

Figura 28: Performance di SipHash in relazione ad altri algoritmi Hash (crittografici e non)

3.4.3: Realizzazione Contromisura HSec

Dalla lunga fase di progettazione per la contromisura emerge l'esigenza di voler costruire un protocollo applicativo e proprietario per poter gestire le diverse difficoltà di programmare una soluzione con le modalità di sicurezza che garantisce BLE: viene proposta una soluzione che in linea teorica dovrebbe soddisfare i requisiti di autenticazione e di non manomissione richiesti: HSEC (Handshake SECurity). Nel dettaglio, questa soluzione è composta da due componenti, con l'obiettivo di andare a mettere in sicurezza la fase di interazione tra Gateway e dispositivo IoT: un protocollo di configurazione tra le due entità (stile MAC) e un semplice diagramma a stati all'interno del dispositivo IoT. Dopo aver presentato dettagliatamente entrambi le parti, e descritto quindi come integrare HSec al prototipo aziendale, verrà valutato in che modo andare a validare questa contromisura nel caso specifico di utilizzo.

3.4.3.1: Protocollo applicativo HSec

La prima parte della soluzione HSec riguarda la progettazione e la realizzazione di un protocollo di interazione che viene affrontato tra Gateway e Arduino non appena è stato effettuato il collegamento (pairing), prima di procedere con qualsiasi altra operazione, come mostrato in figura 29:

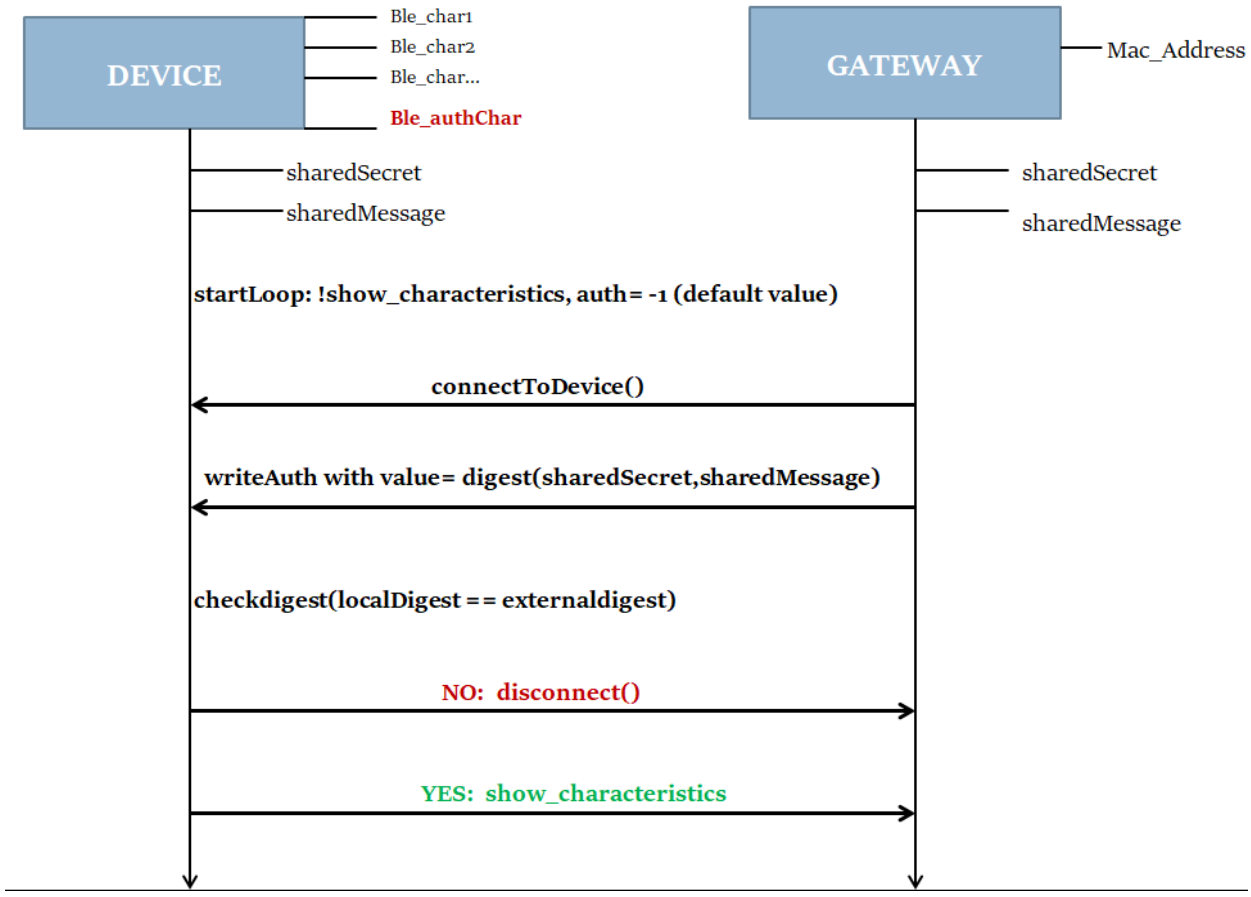

Figura 29: HSec Application Protocol

Questo protocollo potrebbe essere descritto in determinate fasi:

- **Fase 0** : Gateway e Dispositivo vengono connessi in rete. Il Gateway è dotato di uno specifico MAC Address e di due informazioni aggiuntive, *sharedSecret* e *sharedMessage*, con le quali può calcolare attraverso una funzione hash (utilizzando la libreria SipHash) un'impronta (*externalDigest*). Arduino, oltre ad avere un indirizzo MAC, espone i propri dati sottoforma di caratteristiche, e prevede una caratteristica aggiuntiva chiamata *authCharacteristic* (di tipo long e di lunghezza 16 byte), inizializzata a un valore di default; detiene inoltre anche gli stessi *sharedSecret e sharedMessage* che possiede anche il Gateway, scambiati nella fase di cablaggio con esso, e quindi calcola in locale l'impronta su questi due dati in ingresso (*localDigest*).

- **Fase 1**: Arduino inizia il proprio loop, non mostrando (e non permettendo) la modifica delle proprie caratteristiche fino a che chi si collega non si autentica. A questo punto il Gateway, dopo essersi connesso ad esso, va a scrivere sulla caratteristica apposita (*authCharacteristic*) il valore dell'impronta calcolata sui dati in ingresso, attendendo un feedback da Arduino sull'esito della verifica di quest'impronta.

- **Fase 2:** Arduino verifica l'impronta ricevuta sulla caratteristica con l'impronta calcolata in locale, comunicando l'esito di questa verifica (**2.a:** negativo, **2.b:** positivo).

- **Fase 2.a:** Arduino compara le due impronte e non trova una corrispondenza: ciò potrebbe essere dovuto da pacchetti persi in rete o anche da un attacco sulle comunicazioni. In ogni caso, invalida la connessione con chi l'ha trasmessa, tenendo ancora protette le proprie caratteristiche. Arduino invalida la connessione anche nel caso in cui Gateway (o altra entità) va a fare una lettura/scrittura su altre caratteristiche prima di autenticarsi tramite questi passaggi.

- **Fase 2.b:** Arduino compara le due impronte e trova una corrispondenza: questo abilita la lettura/scrittura verso le altre caratteristiche, registrando il MAC del Gateway come indirizzo "fidato", per le prossime richieste di connessioni.

- **Fase 3.b:** Arduino e Gateway fidato si disconnettono: alla prossima richiesta di connessione, Arduino verifica se il MAC della richiesta è lo stesso della precedente iterazione: in caso affermativo, salta il protocollo di autenticazione e diventa subito operativo, dato che la richiesta arriva dallo stesso MAC registrato come fidato. Nel caso sia una richiesta di un altro dispositivo (indirizzo MAC diverso), Arduino protegge le proprie caratteristiche e richiede una nuova iterazione del processo di autenticazione.

Ci sono da svolgere alcune considerazioni. In particolare, questo schema è molto simile allo stile MAC di autenticazione, a meno di un messaggio iniziale (*sharedMessage*) che è stato precablato anziché trasmesso in chiaro in rete come accade nel protocollo MAC: questa scelta è stata effettuata nell'ottica di calcolare immediatamente l'impronta locale nella fase di setup di arduino, risparmiando risorse computazionali rispetto a un'eventuale calcolo "run-time": non appena arriva l'impronta, andiamo solamente a fare un confronto tra due informazioni senza fare ulteriori calcoli. L'utilizzo di SipHash, oltre a garantire autenticità e integrità del messaggio relativo all'impronta, ci permette di ottimizzare di gran lunga la computazione e soprattutto il calcolo dell'impronta lato arduino, garantendo migliori performance. E' importante infine notare che per motivi tecnologici (limite della libreria *gatttool*) è possibile solamente scrivere 8 byte alla volta in una caratteristica, rispetto ai 16 byte necessari per strutturare l'impronta. Per questi motivi, la seconda parte della soluzione è essenzialmente la progettazione di un diagramma a stati che dovrà far parte del dispositivo IoT di riferimento (nel caso specifico, Arduino). Si descrive in dettaglio nella seconda parte anche lo scenario 3.b (che riguarda la gestione dell'indirizzo MAC fidato).

3.4.3.2: Macchina a stati HSEC

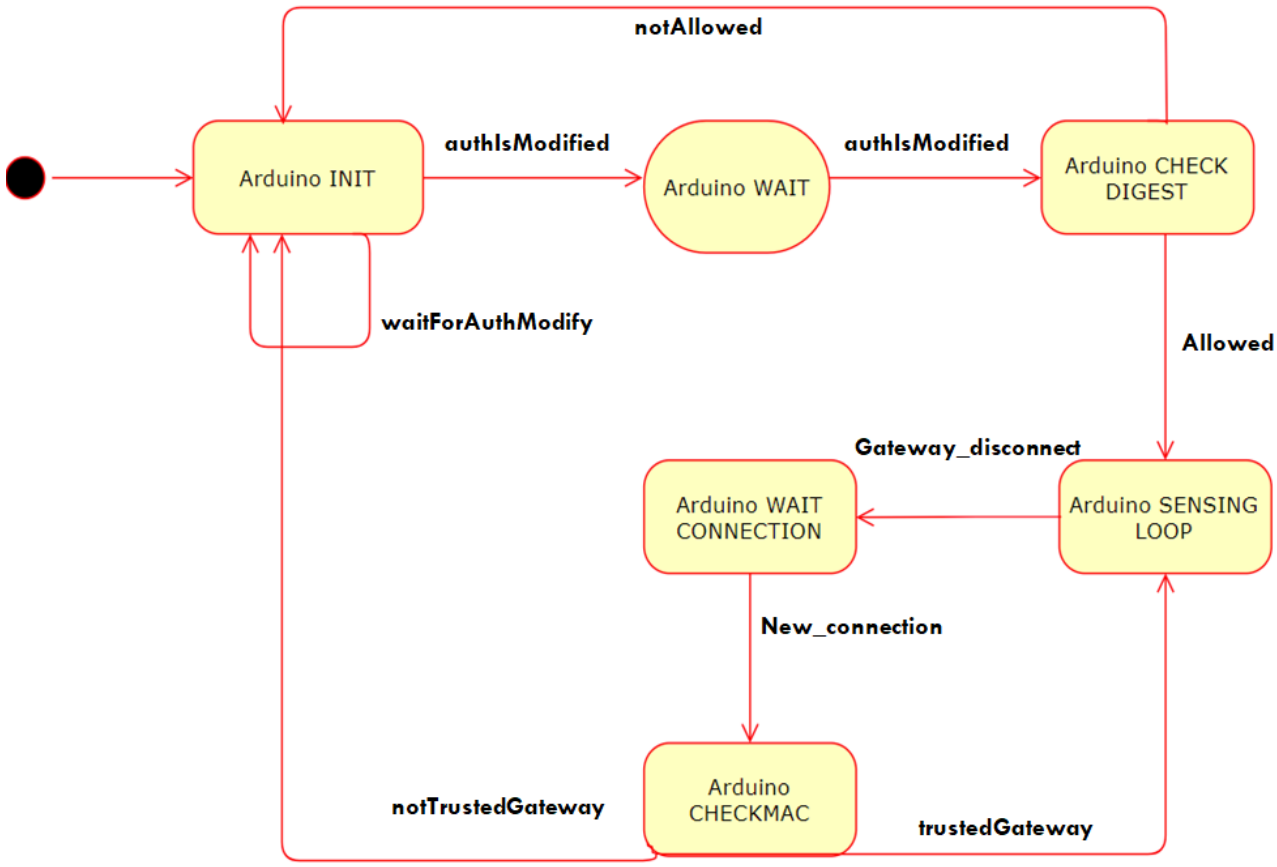

Figura 30: HSec State Machine

Il diagramma a stati a cui si fa riferimento in figura 30 dovrà essere realizzato all'interno del dispositivo IoT viene progettato tenendo conto dei limiti tecnologici dell'interazione con Arduino (scrittura di 8 byte alla volta) e del flusso che il dispositivo stesso deve gestire nell'interazione con il Gateway per il protocollo di autenticazione. Questo diagramma rappresenta il ciclo di vita del dispositivo IoT (Arduino) dopo aver calcolato l'impronta in locale e dopo aver iniziato una connessione con un'entità esterna.

In particolare:

- **Arduino INIT:** Arduino protegge le proprie caratteristiche che sono visualizzate a un valore di default (ad esempio: *authCharacteristic* = -1) e non sono editabili. Se altre caratteristiche vengono editate prima di quest'ultima, Arduino si disconnette. Rimane in questo stato fino a che la caratteristica *authCharacteristic* viene scritta dall'esterno per la prima volta*(authIsModified).*

- **Arduino WAIT:** Arduino memorizza il valore che rileva in *authCharacteristic* (sono i primi 8 byte dell'impronta), imposta un nuovo valore di default per la caratteristica (*authCharacterstic* = 0) e continua a proteggere tutte le caratteristiche. Se qualcuno prova a editare le altre caratteristiche, Arduino si disconnette. Rimane in questo stato fino a che la caratteristica *authCharacteristic* viene cambiata nuovamente *(authIsModified).*

- **Arduino CHECK DIGEST:** Arduino memorizza il valore che rileva in *authCharacteristic* (sono gli ultimi 8 byte dell'impronta), concatena i due valori memorizzati e controlla se l'impronta in locale combacia con l'impronta esterna. In caso positivo (*allowed),* Arduino va a sbloccare le proprie caratteristiche e registra il MAC come dispositivo fidato. In caso negativo, Arduino torna nello stato Init (*notAllowed),* segnalando il problema riscontrato ed eventualmente disconnettendosi.

- **Arduino SENSING LOOP:** Arduino è in loop e sta comunicando con il Gateway fidato: permette ogni tipo di lettura/scrittura. Esce dallo stato solo a seguito di una disconnessione (*gateway_disconnect*).

- **Arduino WAIT CONNECTION:** Arduino attende una nuova connessione: protegge le proprie caratteristiche come nello stato INIT. Non appena si collega con una nuova entità, va nello stato CHECK MAC (*new_connection*).

- **Arduino CHECK MAC:** Arduino controlla se il MAC del dispositivo connesso è uno dei dispositivi fidati: in caso positivo, va direttamente nel SENSING LOOP. In caso negativo, torna nello stato INIT, forzando il nuovo dispositivo all'autenticazione.

Alcune considerazioni: ad *authCharacteristic* vengono impostati due valori di default diversi perché occorre discriminare due stati, ovvero quando la caratteristica dev'essere scritta per la prima volta (-1) e quando essa dev'essere scritta per la seconda volta (0). Inoltre, lo stato CHECK MAC viene inserito per evitare che il Gateway fidato vada a rifare più volte lo stesso protocollo di autenticazione per poter instaurare altre connessioni con il dispositivo: unica pecca di questa facilitazione è rappresentata dalla possibilità che un attacco di **mac spoofing** può andare a invalidare questa contromisura. Si discuterà accuratamente nella parte finale dell'elaborato perché è plausibile che questo problema si presenti, come esso è stato, può (e deve) essere affrontato, ed infine come inserirlo e prevederlo nella soluzione proposta dalla metodologia presentata.

3.4.4: Validazione HSec

Una volta progettata e realizzata la contromisura, occorre capire come validare e controllare che essa garantisca i requisiti di autenticazione e di non manomissione richiesti: non solo, nell'ottica di voler automatizzare nelle prossime iterazioni l'intera fase di ricerca di vulnerabilità, occorre progettare una vera e propria soluzione di validazione che possa essere automatizzata e possa fare riferimento a veri e propri *Security Acceptance Tests* che si inseriranno della pipeline di CD, facilitando in questo modo l'inserimento della sicurezza nel processo di sviluppo di software "by design", completando il ciclo intrapreso con la metodologia delle *abuser stories*.

Come validare la contromisura applicativa proposta? La validazione deve passare per l'effettiva interazione con l'architettura IoT, e soprattutto dovrà tenere conto di verificare se questo protocollo di autenticazione sia stabile oppure no. Nel mondo della sicurezza informatica esiste un vero e proprio filone tecnologico che ha creato nel mondo del lavoro persino una nuova figura esperta, ovvero quella dell'*hacker etico*[WIK1]: la tecnica di penetration testing [ROU, 2011]. Già descritti nel capitolo precedente mettendo in risalto pro e contro generali, i penetration test possono essere automatizzati tramite applicazioni software (e nell' ottica di gestire tantissimi dati, questa potrebbe essere una scelta giusta) oppure performati manualmente: il principale obiettivo di questi test è quello di determinare le debolezze di sicurezza in un sistema. In particolare si hanno due tipologie di penetration testing: interno ed esterno [SAW, 2008]. Nel pentest esterno si va a provare se un'attaccante esterno alla rete, senza nessuna conoscenza interna del sistema ma con la possibilità di utilizzare degli access point più o meno visibili, può condurre attacchi al suo interno: nel penetration interno si va invece a provare che cosa potrebbe fare un attaccante una volta all'interno della rete, e quindi in qualche modo con maggiori accessi alle informazioni rispetto al pentest esterno. Potenzialmente più dannoso date le maggiori possibilità di un attaccante, il pentest interno potrebbe essere però maggiormente vicino al caso di studio in questione, dato che la soluzione è ancora in una fase prototipale (e soprattutto data l'impossibilità di Arduino di connettersi direttamente via Web). A questo punto si potrebbe valutare qual è lo stato dell'arte in materia di soluzioni di pentest che vanno a verificare l'efficacia del protocollo applicativo inserito (provando scenari con e senza questa contromisura), e soprattutto sull'eventuale possibilità di non manomissione di un dispositivo fisico. Come già descritto, Kali propone una lista molto lunga di tool per effettuare penetration test, e nella ricerca occorre tenere in mente i requisiti che vogliamo garantire al fine di provare un piano di attacco che li verifichi: in particolare, occorre decidere se

cercare un tool che vada ad intercettare il pacchetto BLE in rete lasciando ai developer il compito di fare delle verifiche su questa informazione, oppure cercare di progettare un test con l'utilizzo di uno o più tool per provare in automatico la manomissione del dispositivo e dei relativi dati sui sensori, discutendone l'esito. Non si vuole condurre un pentest esterno sull'architettura, dato lo stato prototipale della soluzione, ma si vogliono cercare degli strumenti per fare intercettazione di informazioni (*information gathering*). Tra le diverse soluzioni presenti in Kali si sono andate a discutere quelle che si avvicinano maggiormente al nostro obiettivo: i tool Metasploit, BlueLog, BlueMaho, WireShark e l'utilizzo di uno sniffer Texas Instrument. L'utilizzo di questi tool permette in qualche modo di simulare la fase di raccolta di informazioni sulle comunicazioni, ma ogni soluzione presenta dei limiti che bloccano la progettazione di una semplice verifica applicativa. In particolare, MetaSploit è uno dei più tool più famosi per simulare attacchi passivi su una comunicazione, con il limite d'altronde che lavora solo su protocollo IP (Arduino non va in rete); BlueLog riesce a lavorare solamente con dispositivi molto vicini a lui, BlueMaho nonostante il complesso parco di test di sicurezza che presenta non è trasparente al suo utilizzo e rimanda al programmatore la scelta dei tool da integrare al suo flusso, ed infine WireShark e lo sniffer Texas Instrument hanno entrambi un problema già ampiamente definito, ovvero la difficoltà di gestione del frequency hopping, obbligando inoltre l'utilizzo di hardware dedicato (USB Dongle). Condurre una validazione di un protocollo che è stato inserito a livello applicativo, quindi, risulta essere molto complicato utilizzando i test di penetrazione che sono stati invece progettati e realizzati per verifica di scenari molto più complessi e molto più legati al livello di rete. Oltre a questo rischio, è importante evidenziare la difficoltà di utilizzo di Kali come sistema operativo *as is* a causa di un ridotto numero di strumenti computazionali all'interno dell'azienda e la difficoltà nel virtualizzare nuovi ambienti dovuta a questa limitata strumentazione: questo problema, in realtà, è ampiamente superabile grazie all'utilizzo di un CAAS (Container As A Service) come Docker. Docker[DOC] è un progetto open-source che automatizza il deployment (consegna e rilascio al cliente), con la relativa installazione e messa in funzione di un'applicazione o di un sistema software all'interno di container software, intesi come immagini di sistemi operativi e driver dedicati, fornendo un'astrazione grazie alla virtualizzazione a livello di sistema operativo. In pochissime parole, Docker rappresenta un'alternativa molto più leggera e funzionale all'utilizzo di macchine virtuali, appoggiandosi sul Kernel Linux: si veda figura 31. L'appoggio di Kali Linux sopra questo servizio, dunque, risolve le diverse problematiche legate alle risorse computazionali fisiche e alla difficoltà di virtualizzare questo sistema operativo (superando i problemi legati ai driver).

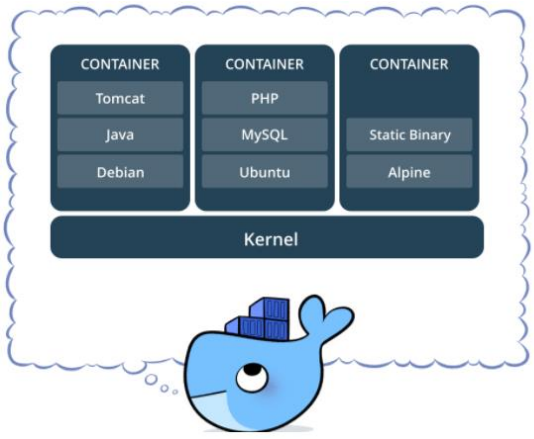

Figura 31: Soluzione Docker

Anche se con Docker viene superato abbastanza agevolmente il problema dell'utilizzo all'interno dell'azienda di Kali Linux come ambiente di testing, i limiti rappresentati dall'inesistenza di tool specifici per andare a validare contromisure applicative e di non manomissione bloccano la progettazione di un vero e proprio piano d'attacco con metodologia penetration. Per questo motivo si è deciso di affrontare la validazione con un'altra scelta progettuale: non potendo riutilizzare come già descritto nel capitolo precedente lo stesso framework che ha evidenziato la vulnerabilità (Gattacker), viene affrontato il percorso di validazione della contromisura attraverso degli scenari reali. Per scenari reali si intende, immaginando che questa soluzione di autenticazione prototipale un giorno possa essere la base (ad esempio) per un sistema reale di login, il test di possibili interazioni che potrebbero esserci tra Gateway e Arduino, e la verifica della contromisura applicativa osservando se l'Arduino, in differenti scenari di interazione con un'altra entità, va effettivamente a proteggere o meno le proprie caratteristiche a seconda dell'esito del protocollo applicativo. Provare scenari reali non risulta complesso, data l'infrastruttura già realizzata nella prima fase di progetto: occorre ragionare sugli scenari possibili da modellare, consapevoli che ovviamente il lavoro non finisce qua (è impossibile modellare ogni scenario reale, e nello stesso momento impossibile determinare preventivamente cosa possa succedere una volta collegato Arduino in "rete").

3.4.4.1: Progettazione scenari reali

Si decide quindi di progettare alcuni scenari reali per andare a validare la contromisura proposta all'interno del sistema. In particolare, vengono definiti scenari che vengono ritenuti significativi sulla realtà, in relazione alla possibile realizzazione successiva di un'interfaccia grafica con cui questo schema di autenticazione sarà presentato:

SCENARIO	Autenticazione HSec	Corrispettiva GUI
Fidato	Scrittura di una caratteristica protetta dopo aver svolto il corretto schema di autenticazione (il *digest* di autenticazione viene mandato corretto come prima operazione)	Login esatto
Autenticazione Errata	Scrittura di una caratteristica protetta dopo aver svolto erroneamente lo schema di autenticazione (in particolare: il *digest* viene mandato in maniera errata)	Login errato
Nessuna Autenticazione	Scrittura di una caratteristica protetta senza effettuare lo schema di autenticazione (si va a scrivere direttamente una caratteristica protetta)	Non effettuare Login

Tabella 8: Differenti scenari reali di validazione

In particolare si vuole andare a testare il comportamento dello schema di autenticazione (e quindi dal punto di vista di Arduino) nel momento in cui un'entità esterna (Gateway, smartphone o altri) va a scrivere su un dato protetto, che nel prototipo specifico è rappresentato dal range di accelerazione (*arange*), che rappresenta una caratteristica sensibile in quanto, se settata su valori non validi, bloccherebbe il flusso di esecuzione dell'Arduino stesso. Analogamente questo discorso potrebbe valere anche per altre caratteristiche sensibili in altri ambiti, come dati personali biometrici, allarmi, antifurti, sensori per pacemaker…

I tre scenari di interazione descritti in figura 32 vengono effettivamente provati attraverso diversi progetti su diversi gateway: i risultati sono descritti in tabella 9 (si rimanda all'appendice per la struttura del diagramma delle classi).

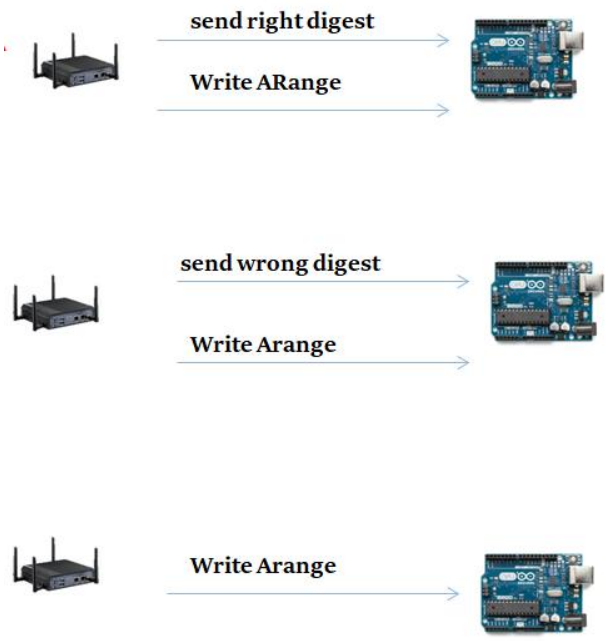

Figura 32: Schema degli scenari di validazione Gateway/Arduino

SCENARIO	Prima di HSec	Dopo HSec
Fidato	V	V
Autenticazione Errata	V	X
Nessuna autenticazione	V	X
Prova MAC	V(stesso / diverso MAC)	V (stesso MAC) / X (MAC diverso)

Tabella 9: Esito della scrittura della caratteristica arange prima e dopo HSec

Dopo l'introduzione di HSec come protocollo di riferimento per l'autenticazione, gli scenari reali vengono modellati garantendo il requisito di autenticazione e di non manomissione: in particolare, viene verificato come ultimo test (Prova MAC) anche che il dispositivo con MAC "fidato" non debba ripetere un'altra volta il processo di autenticazione ma che vada direttamente a dialogare con Arduino. Questo garantisce una facilitazione ma allo stesso tempo una grossa vulnerabilità, che viene subito rilevata e discussa poi nella manutenibilità di questa soluzione nel momento in cui verrà effettivamente automatizzata e inserita nella pipeline di CD, associandola ad eventuali altri flussi di *abuser stories* e in particolare a strumenti di Machine Learning per gestire questa problematica.

I test relativi all'abuser story definita ad inizio progetto sono stati dunque validati e verificati con questa contromisura, in maniera tale che entrambe le fasi (di verifica della vulnerabilità e di realizzazione della contromisura) siano totalmente orientate all'automatizzazione: a questo punto, si può considerare l'intera funzionalità di interazione completata. Ovviamente è possibile aggiungere nuovi requisiti di sicurezza a questa interazione (e, quindi, progettare nuove abuser stories), e inoltre si vedrà come sia importante mantenere questa soluzione, ovvero fare in modo che sia in grado di evolversi e di riscontrare nuove "vulnerabilità" run-time, una sfida che supera totalmente il concetto di *security by design.* In realtà, collegandoci alla tematica concreta del MAC Spoofing su questa soluzione, si mostrerà come sia possibile attraverso lo stesso approccio abuser stories integrare strumenti in grado di lavorare sulla manutenibilità della soluzione (in particolare, che utilizzano tecniche di machine learning per la gestione di grandi quantità di dati e rilevazione di traffico "malevolo").

3.4.5: Manutenibilità HSec

In questo paragrafo si illustra brevemente in maniera teorica come può essere affrontato il problema di mantenere una soluzione che è stata progettata attraverso la metodologia *abuser stories*, come è stato fatto con HSec, presentando gli strumenti e il flusso di lavoro per poter integrare in questa metodologia soluzioni anche legate a mondi più complessi, come quelli del Machine Learning, tenendo presente che è importante gestire il limite evidente che è rappresentato dall' impossibilità di determinare contromisure difensive durante il funzionamento del sistema software stesso, ovvero che non sono state stabilite "by design".

HSec presenta una vulnerabilità nel diagramma a stati di Arduino, che potrebbe essere anche riscontrata run-time ma senza dubbio è visibile già dalla fase di progettazione: nel momento della facilitazione per l'indirizzo MAC fidato, se l'avversario è in grado di "mascherare" il proprio MAC e clonare quello fidato (attacco di MAC Spoofing), Arduino non si accorge e l'avversario salta la fase di autenticazione, violando il protocollo appena costruito. Il MAC Spoofing fa riferimento ad un meccanismo che un dispositivo può realizzare per poter modificare il proprio indirizzo MAC, cercandolo di "mascherare" con un altro indirizzo: anche se molte volte questa tecnica non è sinonimo di attacco informatico, in tantissimi casi un avversario potrebbe voler modificare il proprio MAC per violare meccanismi di accesso e autorizzazione in determinate reti che

controllano gli accessi con gli indirizzi MAC: l'errore più grossolano è infatti presupporre che il MAC sia un identificativo univoco, data la facilità con cui può essere replicato. Definito perché un avversario voglia clonare il MAC, in HSec risulta evidente che la clonazione del MAC potrebbe permettere ad un avversario di spacciarsi per l'entità fidata e autorizzata, senza alcun controllo da parte di Arduino.

Ecco quindi che una nuova *abuser story* potrebbe essere definita:

- **AS A**: avversario
- **I WANT TO:** clonare indirizzo MAC gateway fidato
- **SO THAT:** violare la contromisura HSec, spacciandomi per l'entità autorizzata precedentemente

Un nuovo test associato alla storia può essenzialmente riguardare la prova effettiva di clonazione del MAC e di violazione dello schema di autenticazione HSec, sulla falsa riga di ciò che è stato progettato precedentemente: la novità consisterebbe nel meccanismo con cui è possibile implementare una contromisura a questo scenario.

Tra le tante tematiche che sono discusse quando si parla di sicurezza in ambito IoT una tra tutte è la possibilità di inserire strumenti di analitica Machine Learning per la gestione e la scoperta *smart* di eventuali vulnerabilità e rilevazione di minacce. E' da evidenziare però come il Machine Learning sia molto complicato da associare ad una soluzione di sicurezza: con Machine Learning viene inteso l'insieme di tecniche e algoritmi in grado di apprendere autonomamente tramite dei dati dei pattern, delle caratteristiche e delle correlazioni, in maniera tale che questo apprendimento vada a modificare la natura stessa del funzionamento di questi algoritmi, al fine di portare un risultato. La complessità è determinata dalla difficoltà di reperibilità di dati in grado di "allenare" questi algoritmi (basti pensare ai dati "sporchi" dovuti dall'instabilità della rete), dalla difficoltà di scelta nell'utilizzo di una tecnica o di un'altra (il Machine Learning deve essere direzionato verso un attacco e una configurazione di sicurezza precisa e specifica), e dalla ancora instabilità delle soluzioni attuali (i falsi positivi sono un problema molto grande, collegato anche ad un grande costo [DAN, 2017]). In sostanza, il Machine Learning può essere di aiuto per verificare se ci sono alcune attività anomale in rete (ad esempio, nella funzione di Network Intrusion Detection), ma le decisioni finali spettano sempre agli analisti o agli sviluppatori che si occupano di sicurezza. [ANS, 2017]

Nel caso di HSec, è possibile inserire una soluzione di analitica Machine Learning proprio per andare a testare l'attacco di Mac Spoofing: in particolare, può essere progettato in laboratorio un test in grado di raccogliere un numero molto grande di dati sia del dispositivo legittimo che di quello malevolo, considerando un valore specifico chiamato RSS, e vengono applicate soluzioni di Machine Learning (di cluster per il mondo non supervisionato, e di classificazione per quanto riguarda il mondo supervisionato), in grado di stabilire a run-time se siamo nello scenario di un MAC Spoofing oppure no: con questo sistema si riesce nello stesso momento sia a implementare una contromisura, sia a testare che la storia non possa più essere verificata (anche se l'avversario clonasse il MAC, l'Arduino riuscirebbe a rilevare un problema), e nello stesso tempo automatizzata: in questo modo si riesce a mantenere la soluzione HSec anche per scenari più complessi e rilevabili "run-time", riapplicando questa metodologia in maniera iterativa e collegando senza particolari problemi strumenti di Machine Learning ad-hoc.

Per la spiegazione più approfondita del parametro RSS e dello schema preciso di Machine Learning che può essere utilizzato, si rimanda all'appendice.

Conclusioni

In questo lavoro di Tesi svolto presso il reparto R&D dell'azienda Scai Consulting si è andato a discutere, progettare, realizzare e mantenere una soluzione di sicurezza per un prototipo IoT aziendale.

In particolare, è stata realizzata una soluzione per garantire il requisito di autenticazione e di non manomissione di un dispositivo IoT nella sua interazione con la rete e il mondo esterno, andando ad applicare un flusso di lavoro orientato alla piena automatizzazione e al mondo Agile.

Applicando una particolare metodologia di lavoro (*abuser stories*) si è cercato di verificare se la realizzazione di una contromisura di sicurezza con questo approccio sia effettivamente testabile, estendibile e mantenibile.

La metodologia *abuser stories* in particolare necessita di un flusso di lavoro molto preciso: definizione dei requisiti di sicurezza da proteggere mettendosi dal punto di vista di un attaccante, progettazione e realizzazione dei test associati alla verifica della vulnerabilità descritta dall'attacco, valutazione delle conseguenze della ricerca della vulnerabilità, progettazione e realizzazione di un'eventuale contromisura, validazione della contromisura e automatizzazione delle varie fasi di verifica, con successiva manutenibilità della soluzione.

Nel lavoro di Tesi è stato applicato un flusso preciso per garantire il requisito di autenticazione e di non manomissione del dispositivo IoT. Dopo aver definito formalmente i test da verificare, sono stati realizzati formalmente gli attacchi di ascolto passivo e manomissione attraverso un framework per BLE chiamato Gattacker: è stata progettata e realizzata una soluzione applicativa che garantisse il requisito di sicurezza descritto (HSec), validandola attraverso degli scenari reali e valutando un'eventuale inserimento di analitiche relative al mondo del Machine Learning per la gestione della rilevazione di un attacco di Mac Spoofing durante la vita di questa soluzione (Random Forest e Manutenibilità).

Per quanto occorra in futuro andare a delineare lo stesso ciclo di lavoro per altri requisiti di sicurezza (come ad esempio confidenzialità, integrità, riservatezza), e per quanto occorra automatizzare le varie operazioni di verifica di vulnerabilità, i risultati hanno verificato che questo approccio garantisce una soluzione efficace per la messa in sicurezza di requisiti specificati in fase di analisi, andando ad affrontare e risolvere il problema della mancanza di uno standard univoco di sicurezza in IoT e della difficoltà nel trovare soluzioni definitive a causa dell'eterogeneità e bassa potenza computazionale dei dispositivi, proponendo invece un flusso di lavoro pienamente usabile, estendibile e soprattutto mantenibile e integrabile con il ciclo di sviluppo software Agile.

Per poter definire una vera e propria soluzione completa per la sicurezza non basta andare a verificare durante la vita del sistema software che i requisiti progettati by design siano verificati sempre (principio che è garantito dall'automatizzazione e dall'inserimento dei test di *abuser stories* nella pipeline di CD, prima dei rilasci), ma occorre definire un piano di azione per potere verificare se runtime è possibile che qualche scenario inatteso e malevolo si verifichi: ecco l'importanza dell'estendere questo progetto verso una manutenibilità che renderebbe la soluzione IoT-sicura.

È stato verificato nei punti finale di progettazione che l'integrazione di tecniche in grado di gestire una grande quantità di dati può avvenire senza troppo overhead in questa metodologia, e la ri-applicazione di una nuova iterazione della metodologia può garantire senza troppi costi aggiuntivi una nuova verifica e un nuovo flusso di test, pienamente integrabili con la soluzione.

Con la parte di manutenibilità della soluzione tramite strumenti di analitica *smart* si è andato a delineare un processo completo di messa in sicurezza di un aspetto critico per quanto riguarda l'IoT, ovvero quello dell'autenticazione sul dispositivo, che con un sistema di autorizzazione sulla rete (controllo degli accessi, definizione di utenti, ruoli e azioni) va esattamente a concretizzare uno schema di riferimento teorico che Cisco ha realizzato per presentare un framework di sicurezza IoT/M2M per l'autenticazione, come mostrato in figura 32.

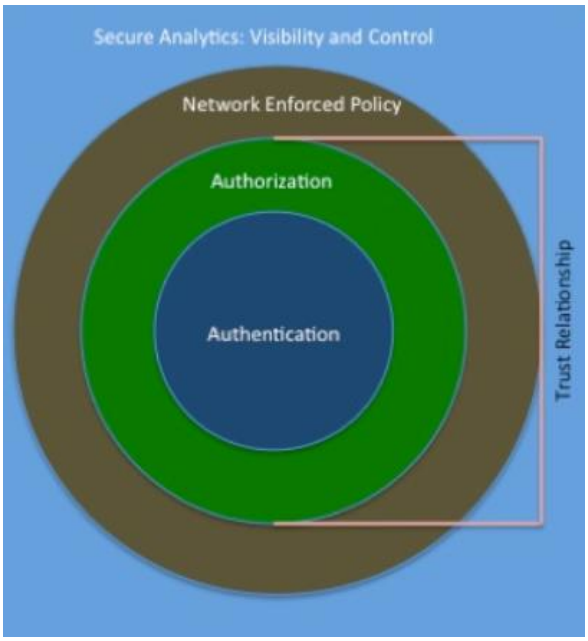

Figura 32: Schema per sicurezza in termini di autenticazione IoT/M2M (Cisco)

Nel lavoro di Tesi sono stati delineati concretamente i livelli che nello schema di Cisco fanno riferimento ad Authentication (con Abuser Stories) e in una prima parte di Secure Analytics (con Machine Learning).

Questo lavoro di Tesi presenta un primissimo approccio e soluzione prototipale e pioneristica verso un problema e una tematica che in questo momento è in pieno fermento: per questo motivo alcuni dei punti potrebbero risultare essere poco chiari o comunque non in grado di coprire tutti gli scenari possibili.

In particolare, l'approccio *abuser stories* si integra molto bene con la progettazione di soluzioni IoT prototipali, in realtà aziendali in cui è possibile avvalersi di figure esperte per quanto riguarda l'analisi di sicurezza (in grado quindi di parallelizzare il lavoro che altrimenti risulterebbe macchinoso), anche se è possibile integrarlo pure in un contesto di sviluppo che manca di un

background diretto di esperienza sulla sicurezza. Risulta invece inapplicabile o comunque molto complicato da applicare in sistemi già “consolidati” e non in fase di prototipo, a causa della difficoltà nel provare scenari che sono già vere e proprie funzionalità di sistema, eventualmente composte e interconnesse tra loro.

Appendice

I- Progetto Software HSec

In questa appendice vengono presentati alcuni punti di dettaglio tecnologico che sono stati affrontati ed implementati durante il progetto di ricerca. Infine si propone una roadmap per delineare le attività che l'azienda dovrà intraprendere da gennaio 2018 per consolidare la soluzione specifica HSec (e, in generale, una soluzione di sicurezza per IoT) nel corso del nuovo anno.

Codice framework Gattacker

File json:

```
[...]
 {
        "uuid": "19b10012e8f2537e4f6cd104768a1214",
        "name": null,
        "properties": [
          "read",
          "notify"
        ],
        "value": "0000b4bb",
        "descriptors": [
          {
            "handle": 14,
            "uuid": "2902",
            "value": ""
          }
        ],
        "startHandle": 12,
        "valueHandle": 13,
        "asciiValue": "    ",

                "hooks":{
                        "dynamicRead": "hookTest"
                }
    },
[...]
```

Implementazione test applicativi (NodeJS):

```
var fs = require('fs');
var colors = require('colors');
var utils = require('../lib/utils');
var actNotify = false;
var actWrite = '';
var operationToTest='dynamicEavesDrop';

var functionMapper = {
	'dynamicEavesDrop': dynamicEavesDrop,
	'dynamicTampering': dynamicTampering
}

function hookForTestingFunction(peripheralId, service, characteristic, type, data, eventEmitter, callback){

	console.log('Hello there');
	functionMapper[operationToTest](peripheralId, service, characteristic, type, data, eventEmitter, callback);
}

function dynamicEavesDrop (peripheralId, service, characteristic, type, data, eventEmitter,callback){
	console.log('I ve just intercept your read operation' , peripheralId);
	console.log('Here it is the service value: ' , service);
	console.log('Here it is the characteristic' , characteristic);
	console.log('Type is: ' , type);
	console.log('Data :' , data);
	//console.log('Event emitter: ' , eventEmitter);
	callback(null,data);
	return;
}

function dynamicTampering (peripheralId, service, characteristic, type, data, eventEmitter,callback){
	console.log('Hello I will try to tamper the device');
	var m = 0;
	data = m.toString(16);
	callback(null,data);
	return;
}
```

Codice Sketch Arduino

```
#include "CurieIMU.h"
#include "CurieBLE.h"
#include "HexConversionUtils.h"
#include "SipHash_2_4.h"
#include <stdio.h>

int aix, aiy, aiz;
int gix, giy, giz;
bool flag = false;
```

```
bool isAllowedFlag = false;
long localDigest = -1;
char digestHex[17];
long firstHalf;
long secondHalf;
boolean readyForTheSecond = false;
String firstHalfString;
String secondHalfString;
String trustedAddress;
BLEPeripheral blePeripheral;
BLEService sensorService("19B10010-E8F2-537E-4F6C-D104768A1214");
BLEUnsignedLongCharacteristic stepCharacteristic("19B10011-E8F2-537E-4F6C-D104768A1214", BLERead);
BLEFloatCharacteristic axCharacteristic("19B10012-E8F2-537E-4F6C-D104768A1214", BLERead | BLENotify);
BLEFloatCharacteristic ayCharacteristic("19B10013-E8F2-537E-4F6C-D104768A1214", BLERead | BLENotify);
BLEFloatCharacteristic azCharacteristic("19B10014-E8F2-537E-4F6C-D104768A1214", BLERead | BLENotify);
BLEFloatCharacteristic gxCharacteristic("19B10015-E8F2-537E-4F6C-D104768A1214", BLERead | BLENotify);
BLEFloatCharacteristic gyCharacteristic("19B10016-E8F2-537E-4F6C-D104768A1214", BLERead | BLENotify);
BLEFloatCharacteristic gzCharacteristic("19B10017-E8F2-537E-4F6C-D104768A1214", BLERead | BLENotify);
BLEIntCharacteristic shockCharacteristic("19B10018-E8F2-537E-4F6C-D104768A1214", BLERead | BLENotify);
BLEIntCharacteristic modeCharacteristic("19B10019-E8F2-537E-4F6C-D104768A1214", BLERead | BLEWrite);
BLEIntCharacteristic srangeCharacteristic("19B10020-E8F2-537E-4F6C-D104768A1214", BLERead |
BLEWrite);
BLEIntCharacteristic arangeCharacteristic("19B10021-E8F2-537E-4F6C-D104768A1214", BLERead |
BLEWrite);
BLEIntCharacteristic grangeCharacteristic("9B100122-E8F2-537E-4F6C-D104768A1214", BLERead |
BLEWrite);
BLELongCharacteristic authCharacteristic("19B100124-E8F2-537E-4F6C-D104768A1214", BLERead |
BLEWrite);  // auth_characteristi

void setup()
{
  Serial.begin(9600);
 // while (!Serial);    // wait for the serial port to open
 CurieIMU.begin();
 // configure Bluetooth
 blePeripheral.setLocalName("MOOVBIT");
 blePeripheral.setAdvertisedServiceUuid(sensorService.uuid());
 blePeripheral.addAttribute(sensorService);
 blePeripheral.addAttribute(stepCharacteristic);
 blePeripheral.addAttribute(axCharacteristic);
 blePeripheral.addAttribute(ayCharacteristic);
 blePeripheral.addAttribute(azCharacteristic);
 blePeripheral.addAttribute(gxCharacteristic);
 blePeripheral.addAttribute(gyCharacteristic);
 blePeripheral.addAttribute(gzCharacteristic);
 blePeripheral.addAttribute(shockCharacteristic);
 blePeripheral.addAttribute(modeCharacteristic);
 blePeripheral.addAttribute(arangeCharacteristic);
 blePeripheral.addAttribute(srangeCharacteristic);
 blePeripheral.addAttribute(grangeCharacteristic);
 blePeripheral.addAttribute(authCharacteristic);
 // set initial characteristics values
```

```
 stepCharacteristic.setValue(0);
 shockCharacteristic.setValue(0);
 modeCharacteristic.setValue(0);
 arangeCharacteristic.setValue(4);
 srangeCharacteristic.setValue(4);
 grangeCharacteristic.setValue(250);
 authCharacteristic.setValue(-1);
 blePeripheral.begin();
 CurieIMU.setAccelerometerRange(arangeCharacteristic.value());
 CurieIMU.setStepDetectionMode(CURIE_IMU_STEP_MODE_SENSITIVE);
 CurieIMU.setStepCountEnabled(true);
 CurieIMU.setDetectionThreshold(CURIE_IMU_SHOCK, srangeCharacteristic.value() * 500); // 1g = 1000 mg
 CurieIMU.setDetectionDuration(CURIE_IMU_SHOCK, 50); // 50ms or 75ms
 CurieIMU.interrupts(CURIE_IMU_SHOCK);
 CurieIMU.attachInterrupt(eventCallback);
 uint8_t key[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
                  0x18, 0x19, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};
 uint8_t message[] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
                  0x18, 0x19, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e};
 int msgLen = 15;
 sipHash.initFromPROGMEM(key);
 for (int j=0; j<msgLen;j++) {
    sipHash.updateHash((byte)message[j]);
  }
 sipHash.finish(); // result in BigEndian format
 hexToAscii(sipHash.result,8,digestHex,17);
}

void loop()
{
 BLECentral bleCentral = blePeripheral.central();
 if (bleCentral) {
   while(bleCentral.connected()){
       String address = bleCentral.address();
       boolean macClientIsAllowed = checkMac(address, trustedAddress);
       if(isAllowedFlag || macClientIsAllowed){
            if(!flag){
              CurieIMU.readMotionSensor(aix, aiy, aiz, gix, giy, giz);
              flag = true;
            }else{
              if(!axCharacteristic.setValue(convertRawAcceleration(aix))
              || !ayCharacteristic.setValue(convertRawAcceleration(aiy))
              || !azCharacteristic.setValue(convertRawAcceleration(aiz))
              || !gxCharacteristic.setValue(convertRawGyro(gix))
              || !gyCharacteristic.setValue(convertRawGyro(giy))
              || !gzCharacteristic.setValue(convertRawGyro(giz))){
                  // do nothing
                }
              stepCharacteristic.setValue(CurieIMU.getStepCount());
              shockCharacteristic.setValue(0);
              flag = false;
            }
```

```
            delay(1000);
        }
        if(authCharacteristic.written()){
                Serial.println("Auth is modified by: ");
                Serial.println(address);
                if(!readyForTheSecond){
                    firstHalf = authCharacteristic.value();
                    firstHalfString = String(firstHalf, 16);
                    authCharacteristic.setValue(0);
                    readyForTheSecond = true;
                }else {
                    readyForTheSecond = false;
                    secondHalf = authCharacteristic.value();
                    secondHalfString = String(secondHalf, 16);
                   if (checkForAllowing(firstHalfString,secondHalfString)){
                        isAllowedFlag = true;
                        Serial.println("allowed: ");
                        trustedAddress= address;
                        Serial.println(address);
                        Serial.println(axCharacteristic.value());
                    } else {
                     isAllowedFlag = false;
                     Serial.println("not allowed:" );
                     Serial.println(address);
                    }
                }
          }
        if(arangeCharacteristic.written()){
            Serial.println("I write the arangeChar");
            if ((isAllowedFlag == false) && (macClientIsAllowed == false)){
                Serial.println("i'm not allowed to write");
                arangeCharacteristic.setValue(0);
                Serial.println(arangeCharacteristic.value());
            } else {
              Serial.println("i'm allowed to write");
              Serial.println(arangeCharacteristic.value());
            }
        }

       }
 }
 isAllowedFlag = false;
}

boolean checkForAllowing(String firstHalf,String secondHalf){
  String digest = secondHalf + firstHalf;
  Serial.println("External digest is: ");
  Serial.println(digest);
  String localDigest = String(digestHex);
  Serial.println("local digest is: ");
  Serial.println(localDigest);
  return localDigest.equalsIgnoreCase(digest);
```

```
}


boolean checkMac(String newMac, String trustedMac){
  return newMac.equalsIgnoreCase(trustedMac);
}
void eventCallback() {
  if (CurieIMU.getInterruptStatus(CURIE_IMU_SHOCK)) {
    shockCharacteristic.setValue(1);
  }
}
```

Refactoring Sketch Arduino:

Il codice presentato è stato oggetto di un lavoro di analisi e di refactoring durante l'ultimo mese del progetto. In relazione a un'attività di tutoraggio nei confronti di due studenti delle scuole superiori croati (Franko e Karlo) è stata proposta loro una problematica inerente questo codice e la validazione della contromisura HSec. In particolare, da parte loro è stata affrontata l'analisi di una problematica su questo codice: Arduino, sia nel caso di errata autenticazione che nel caso di una non-autenticazione manda lo stesso "segnale" di errore (valore 0 nella caratteristica di autenticazione). Questo comportamento può essere un problema, dato la necessità di discriminare i due casi di errore: in un caso potrebbe esserci un errore che potrebbe essere dovuto dalla rete (errata autenticazione), e quindi occorre lato utente sapere qual è il problema esatto che Arduino ha riscontrato, per poterlo gestire anche un domani con una relativa interfaccia grafica di errore associata, ad esempio. Dopo varie discussioni e illustrato la piccola modifica che occorre fare, si decide dunque di risolvere questo problema discriminando i casi di errore con due valori diversi in uscita.

Progetto Java per Gateway

Viene presentata la struttura delle classi Java che permettono al Gateway della Intel di andare a interagire con Arduino. In particolare viene mostrata la struttura di *HSecProject*, il progetto in cui il Gateway va ad affrontare la fase di autenticazione e infine va ad interagire con Arduino sfruttando *jgatttool* [GIT5, 2017]. Tutte le classi sono state progettate con la metodologia TDD, andando prima a costruire i Test per le varie classi definendo il comportamento by design, andando solo in fase successiva ad implementare le funzionalità associate.

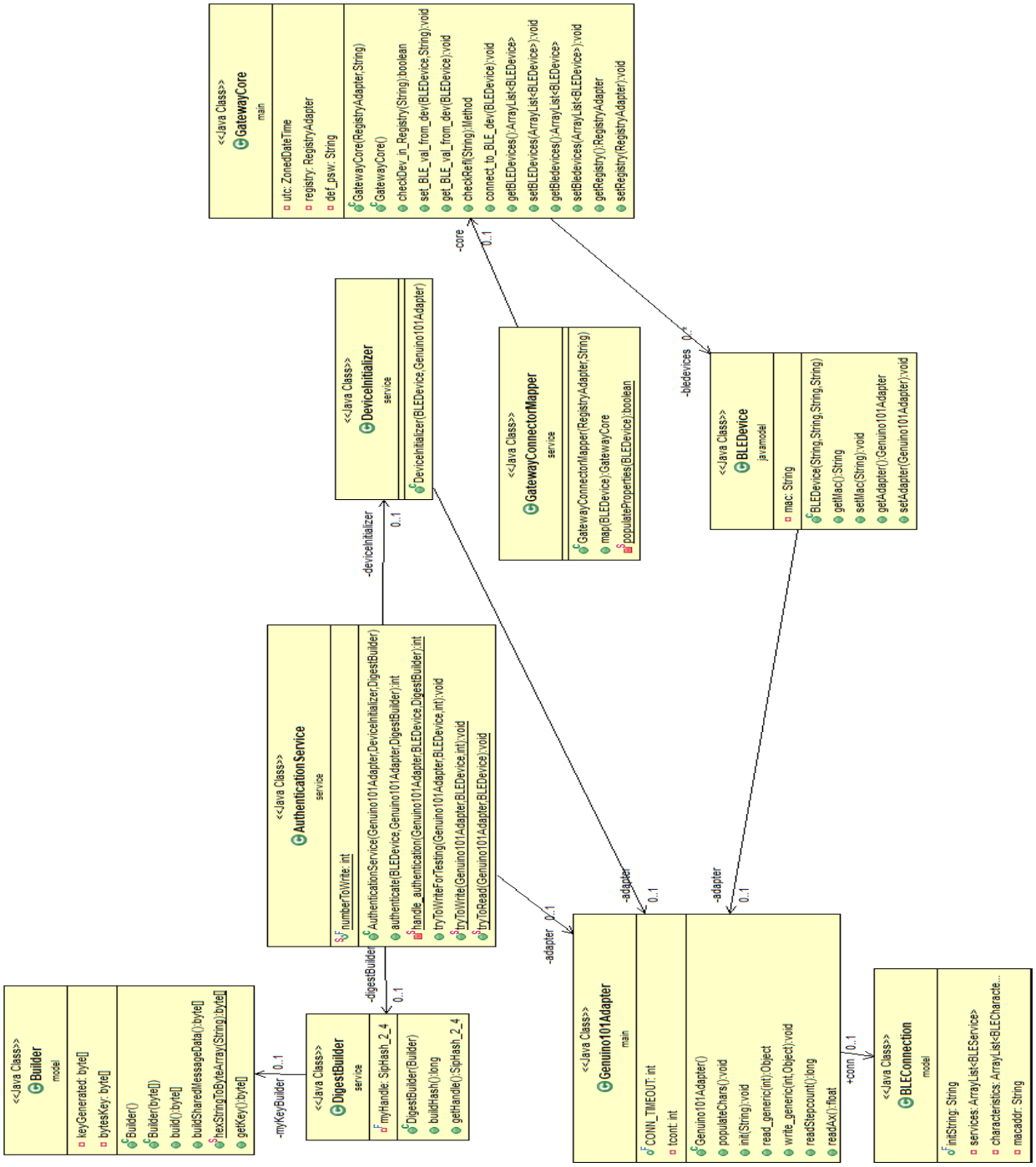

Figura 33: Diagramma classi HSec

Project Roadmap Aziendale per il 2018:

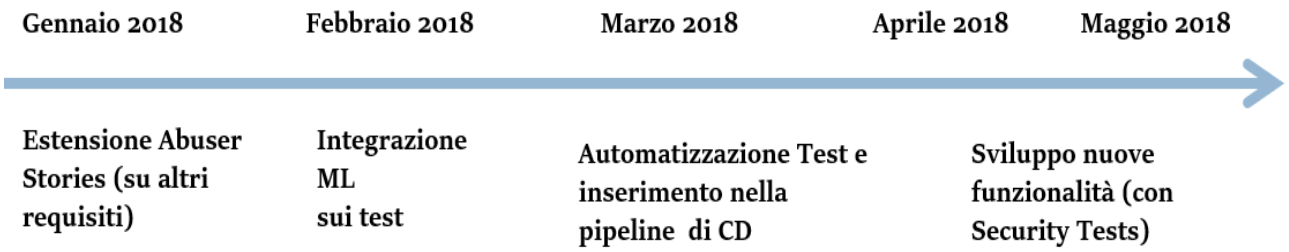

Figura 34: Roadmap di progetto per HSec per il 2018

II- Machine Learning, RSS e Random Forest

Nell'ultima parte del progetto di ricerca è stata svolta una prima analisi verso una problematica molto importante relativa alla soluzione, ovvero quella della manutenibilità della soluzione stessa. Viene in particolare proposto l'utilizzo di tecniche di Machine Learning per andare a verificare, provare e mettere al sicuro un eventuale scenario di MAC Spoofing che potrebbe compromettere la soluzione stessa così com'è stata presentata. Nel dettaglio viene ora spiegato perché per questa tipologia di attacco è necessario affidarsi a un algoritmo di machine learning, e soprattutto come si sono evolute le tecniche per gestire questo scenario: l'intento è quello di mostrare che, dopo aver illustrato un'eventuale progettazione per verificare e provare questo scenario, il collegamento con un nuovo flusso di lavoro (che estende HSec) non risulti essere per niente complicato.

Un avversario, come già spiegato all'interno del trattato, può voler avvalersi della tecnica di clonazione di un indirizzo MAC di un utente legittimo per poter mascherare la propria identità o per aggirare sistemi che danno accesso solamente a indirizzi "fidati", prendendo posto di un utente legittimo e conducendo un attacco nel mezzo (Man in the Middle). Nell'IoT il problema principale consiste proprio nell'andare a determinare in che modo è possibile identificare (autenticare) un dispositivo all'interno della rete, evitando la soluzione banale di identificarlo con il proprio MAC, che è facilmente clonabile. Vengono proposte molte soluzioni in letteratura: l'utilizzo di un timestamp, di un numero di sequenza, tecniche di finterprinting del sistema operativo… tutte tecniche che comunque risultano complicate e nella pratica poco efficaci. Un attacco di MAC Spoofing in IoT può andare a prendere il controllo di sensori sensibili, come pacemakers, telecamere antifurto, e quindi è necessario andare a identificare all'interno della rete dispositivi

malevoli che cercano di effettuare questa operazione. La soluzione proposta [ALO et al., 2016] si basa sull'applicazione di un algoritmo di Random Forest su una particolare tipologia di dato, chiamato RSS.

Con RSS in telecomunicazioni si intende *received signal strength* (RSS), un particolare parametro legato al livello fisico delle comunicazioni che misura la potenza del segnale radio ricevuto. Questo parametro può dipendere da molti fattori, come la distanza tra i dispositivi che comunicano, la potenza con cui il segnale viene trasmesso e la naturale degradazione dell'attrito dell'aria; difficilmente questo segnale cambia durante la vita di un dispositivo, e quindi un'anomala degradazione di questo segnale potrebbe senz'altro far riferimento a un cambiamento del dispositivo stesso che l'ha mandato (e, nella maggior parte dei casi, di un dispositivo che cerca di spacciarsi per il legittimo). Con la forte assunzione che un eventuale dispositivo malevolo sia a una distanza diversa del legittimo, un RSS legato alla ricezione di un indirizzo MAC di un dispositivo degradato rispetto al normale potrebbe essere il segnale di un MAC spoofing. A livello sperimentale sono stati condotti innumerevoli esperimenti sullo scenario: in generale, l'utilizzo di un algoritmo Machine Learning è stato adottato per catalogare tanti campioni di RSS e verificare se runtime veniva riscontrato un valore anomalo rispetto al normale. Sono sempre stati proposti algoritmi di clustering (non supervisionati), con l'obiettivo dunque di fare in modo che l'algoritmo in questione si costruisse runtime i vari cluster e i gruppi in cui catalogare i diversi tipi di dati. Anche se questa soluzione è molto più leggera, l'assunzione forte è sempre stata quella che i dati di RSS avessero una distribuzione gaussiana: la scoperta che questa assunzione non era assolutamente corrispondente a verità (tramite diverse sperimentazioni) ha mostrato come questi algoritmi non supervisionati in realtà fossero poco accurati, mostrando tantissimi casi di falsi positivi [DAN, 2017].

Per questo motivo si va ad illustrare una soluzione basata su un algoritmo di Machine Learning supervisionato e legato al mondo della classificazione e degli alberi decisionali, chiamato Random Forest. Random Forest [BRE et al., 2004] rappresenta un algoritmo che costruisce diversi alberi decisionali: con albero decisionale si intende un classificatore decisionale che, dato un input, costruisce un output e un'etichetta che è basata sulla scelta delle diverse strade all'interno dell'albero, che in una prima fase è costruito su un training set fisso di dati. Per classificare ogni nuovo input, questo viene fatto scendere su tutti gli alberi di decisione della foresta: ogni albero fornisce un responso (etichetta), e la foresta sceglie infine qual è l'etichetta che è stata più "votata". Introducendo la casualità come fattore dominante negli alberi decisionali (ogni albero viene costruito su una porzione casuale di training set, e ogni nodo viene splittato in maniera casuale), questa casualità garantisce una diversità tra i classificatori che rispecchia effettivamente la natura di dati casuali (come, nel nostro caso, i campioni di RSS). Più in particolare, ogni albero cresce in questo modo:

- Se il numero dei casi è N, vengono fatti N campioni random dai dati originali, e questi campioni verranno usati come training set per accrescere l'albero.
- Se ci sono M input, viene stabilito un numero m<<M in maniera tale che in ogni nodo, m variabili sono selezionate random da M e il miglior split su queste variabili viene usato per splittare il nodo. Il valore di m rimane costante per tutta la foresta.

- Non c'è pruning

Questa tecnica garantisce delle caratteristiche:

- È insuperabile in accuratezza tra gli algoritmi correnti.
- Lavora efficientemente su grandi basi di dati.
- Garantisce una stima su quali variabili sono importanti nella classificazione.
- Genera una foresta di alberi decisionali che potrebbe essere molto importante per dati futuri.

La soluzione sperimentale provata in letteratura con questo algoritmo sullo scenario di MAC Spoofing riguarda uno schema che lavora su due fasi: una fase offline in cui il classificatore (console) costruisce il giusto segnale di RSS, e una fase online in cui ogni dato viene giudicato runtime se "malevolo" o no.

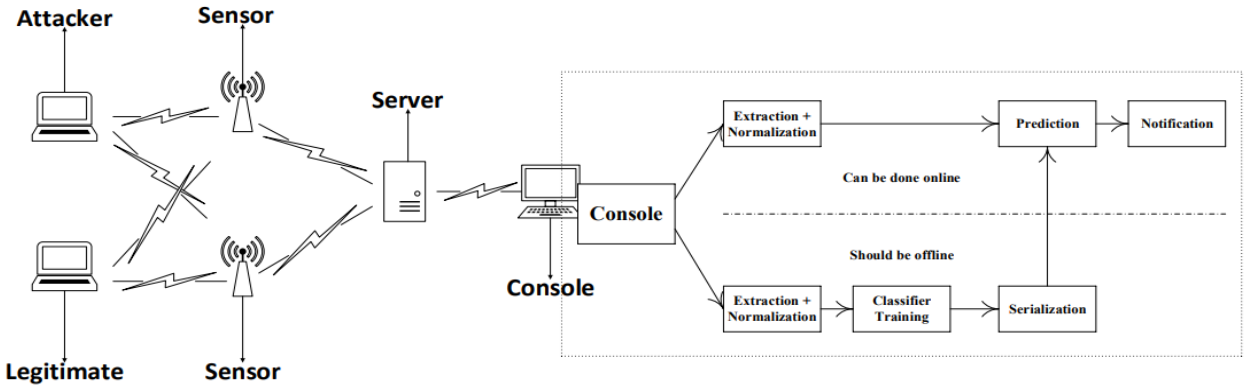

Figura 35: Random Forest di campioni RSS

Questo schema può essere dunque pienamente applicabile allo scenario specifico del prototipo aziendale: nel momento in cui Arduino viene collegato ad un Server (e questo collegamento ovviamente è sicuro), e dopo aver messo in sicurezza il Gateway legittimo, avendone costruito il relativo RSS nella fase offline, nella fase online Arduino (e il server) a cui sarà collegato sarà in grado di gestire un eventuale segnale diverso che potrebbe arrivare da un utente non legittimo, reagendo in maniera preventiva ad un eventuale accesso non autorizzato al sistema, in piena linea con il requisito di manutenibilità e di rilevazione di minaccia "runtime". Il limite sarà quello di

andare a costruire un nuovo flusso di verifica di vulnerabilità e di verifica di contromisura, oltre ovviamente a quello di andare a mettere comunque in sicurezza singolarmente i dispositivi che faranno parte di questo ciclo di test (ad esempio, nel caso in cui *Attacker* prenda possesso di *Legitimate,* lato Server è impossibile da rilevare questa operazione malevola).

È possibile dunque estendere il progetto HSec per integrare questa nuova funzionalità.

Bibliografia

Formato Riferimenti (In ordine alfabetico):

[AUT, ANNO]: "TITOLO", SEDE DI PUBBLICAZIONE, ANNO, NOME DELL'AUTORE(I), (LINK)

[AGW] : "Extreme Programming – Metodologia di sviluppo agile", Agileway, https://www.agileway.it/extreme-programming-metodologia-sviluppo-agile/ ;

[ALO et al. , 2016] : "A new MAC Address Spoofing Detection Technique Based on Random Forests", University Of Bridgeport, 2016, Bandar Alotaibi and Khaled Elleithy;

[ALF et al., 2015] : "Internet Of Things: A survey on Enabling Technologies, Protocols and Applications", IEEE, 2015, Ala Al-Fuqaha et al.;

[ANS, 2017] : "Is The Security Industry Ready for Machine Learning?", SC Media, 2017, Darren Anstee,https://www.scmagazine.com/is-the-security-industry-ready-for-machine-learning/article/685090/ ;

[ARG, 2013] : "Ultimate Guide to Debugging Bluetooth Smart / BLE Products", Argenox , 2013, http://www.argenox.com/bluetooth-low-energy-ble-v4-0-development/library/ultimate-guide-to-debugging-bluetooth-smart-ble-products/ ;

[ARD] : "Arduino Home Page" : https://www.arduino.cc/ ;

[ASA, 2014] : "Why you need NoSQL for the Internet Of Things", Readwrite, 2014, Matt Asay, https://readwrite.com/2014/11/28/internet-of-things-nosql-data/ ;

[AUM et al., 2012] : "SipHash: a fast short-input PRF", Dipartimento di Informatica, Università di Chicago, 2012, Jean-Philippe Aumasson et al.;

[BAN, 2017] : "Three Major Challenges Facing IoT", IEEE Internet Of Things, 2017, Ahmed Banafa, https://iot.ieee.org/newsletter/march-2017/three-major-challenges-facing-iot.html ;

[BEC et al., 2001] : "Manifesto per lo Sviluppo Agile di Software", Agile Manifesto, 2001, Kent Beck et al., http://agilemanifesto.org/iso/it/manifesto.html;

[BEL, 2017] : "Yoroi: la protezione dentro casa supera quella sul posto di lavoro", Itespresso, 2017, Stefano Belviolandi, http://www.itespresso.it/yoroi-la-protezione-dentro-casa-supera-quella-sul-posto-di-lavoro-124992.html?inf_by=59edf91e671db8da018b4704 ;

[BLU, 2000] : "Bluez Official Linux Bluetooth Protocol Stack" , 2000 : http://www.bluez.org/ ;

[BON, 2015] : “A Basic Introduction to BLE Security”, Eewiki, 2015, Matthew Bon, https://eewiki.net/display/Wireless/A+Basic+Introduction+to+BLE+Security ;

[BOW, 2015] : “Kanban vs Scrum vs XP – an Agile comparison”, Manifesto, 2015, Jim Bowes;

[BOW, 2017] : “Wearables: Security Of Things”, Rsaconference, 2017, Marc Bown, https://www.rsaconference.com/blogs/wearables-security-of-things ;

[BRE et al., 2004] : “Random Forests”, Berkeley, 2004, Leo Breiman, Adele Cutler, https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm ;

[BRE, 2016] : “Com’è nato l’Internet of Things? Le 15 date che hanno fatto la storia”, Kiteblue, 2016, Eleonora Breghiroli, http://www.kiteblue.it/come-nato-linternet-of-things-le-15-date-che-hanno-fatto-la-storia-2/#;

[CAQ1, 2016]: “BtleJuice Framework Open-source”, Github , 2016, Damien Cauquil, https://github.com/DigitalSecurity/btlejuice ;

[CAQ2, 2016]: “BtleJuice: The Bluetooth Smart MITM Framework”, Defcon presentation, 2016, Damien Cauquil;

[CAR, 2017] : “Studio e realizzazione di un prototipo di architettura IoT per monitoraggio di atleti in allenamenti sportivi”, Università di Bologna, 2017, Nicolò Carpignoli;

[CAS] : “Apache Cassandra Home Page” : http://cassandra.apache.org/ ;

[CAS,2010] : “Scrum: un processo agile”, MokaByte , 2010, Mirco Casani, http://www.mokabyte.it/2010/09/scrum-1/;

[CAS, 2017] : “Fog Computing: ecco l’evoluzione del cloud pensata per l’Internet Of Things”, Internet4Things Web Library, 2017, Annalisa Casali, https://www.internet4things.it/iot-library/fog-computing-ecco-levoluzione-del-cloud-pensata-per-linternet-of-things/ ;

[CBL] : “CurieBLE library” , Arduino : https://www.arduino.cc/en/Reference/CurieBLE ;

[CRT, 2017] : “Krack Attack: Violato il protocollo di sicurezza WI-FI WPA2”, CERT Nazionale Italia, 2017, https://www.certnazionale.it/news/2017/10/16/krack-attack-violato-il-protocollo-di-sicurezza-wi-fi-wpa2/ ;

[DAN, 2017] : “False Positive Are a True Negative: Using Machine Learning to improve accuracy”, Security Intelligence, 2017, Jack Danahy, https://securityintelligence.com/false-positives-are-a-true-negative-using-machine-learning-to-improve-accuracy/ ;

[DOC] : Docker Documentation: https://www.docker.com/what-docker

[EAS, 2017] : “4 critical security challenges facing IoT”, Networkworld, 2017, Gary Eastwood, https://www.networkworld.com/article/3166106/internet-of-things/4-critical-security-challenges-facing-iot.html ;

[EEN, 2014] : “Is Bluetooth Less Susceptible to jamming than wifi?”, Electrical Engineering StackExchange Blog, 2014, https://electronics.stackexchange.com/questions/121250/is-bluetooth-less-susceptible-to-jamming-than-wifi ;

[FHS] : “Frequency-hopping spread spectrum”, https://en.wikipedia.org/wiki/Frequency-hopping_spread_spectrum ;

[FOR, 2011] : “Penetration Test, analisi di un caso reale” , Html.it, 2011, Raffaele Forte, http://www.html.it/articoli/penetration-test-analisi-di-un-caso-reale-1/ ;

[FOR, 2013] : “Siphash Library For Arduino” : http://www.forward.com.au/pfod/SipHashLibrary/, “Siphash Java Library” : http://www.forward.com.au/pfod/SipHashJavaLibrary/index.html , Forward Computing, 2013, Matthew Ford;

[FRA et al. , 2010] : “Relay Attacks on Passive Keyless Entry and Start Systems in Modern Cars”, Departement of Computer Science Zurich, 2010, Aurelien Francillon et al. ;

[FRE, 2015] : “A reference Architecture for the Internet Of Things”, WSO2 White Paper, 2015, Paul Fremantle;

[GAR, 2016] : “Privacy: nuovo Regolamento Ue, prime Linee guida dei Garanti Europei”, Garante Privacy, 2016, http://www.garanteprivacy.it/web/guest/home/docweb/-/docweb-display/docweb/5792160 ;

[GHI, 2015] : “Auto elettrica: un percorso di sistema”, Webnews, 2015, Cristiano Ghidotti, http://www.webnews.it/2015/09/11/auto-elettrica-sistema/ ;

[GIT1] : “Ubertooth Project open-source”, Github : https://github.com/greatscottgadgets/ubertooth/;

[GIT2] : “Jpcap open-source”, Github: https://github.com/jpcap/jpcap ;

[GIT3] : “Pyshark open-source” , Github : https://github.com/KimiNewt/pyshark ;

[GIT4, 2017] : “Unsupported manufacturer (Qualcomm), Github Issue : https://github.com/securing/gattacker/issues/15;

[GIT5, 2017] : “Jgatttool – Java library for Bluetooth LE GATT” , GitHub, Nicolò Carpignoli, 2017, https://github.com/nicolocarpignoli/jgatttool ;

[GRA et al., 2015] : “Security in the integration of low-power Wireless Sensor Networks with the Internet: A survey”, ScienceDirect, 2015, Jorge Granjal, Edmundo Monteiro, Jorge Sà Silva;

[GRA et al. , 2015] : “Security for the Internet of Things: A Survey of Existing Protocols and Open Research Issues”, IEEE Pubblication , 2015, Jorge Granjal et al. ;

[GUT et al. , 2013] : “The Role of Ad Hoc Networks in the Internet of Things: A Case Scenario for Smart Environments”, ResearchGate, 2013, Daniel Gutierrez et al.;

[HIG, 2002] : “Agile Software Development Ecosystems”, Pearson, 2002, Jim Highsmith;

[HMQ, 2016] : “MQTT Security Fundamentals: TLS/SSL”, HiveMQ Web Portal, 2016, https://www.hivemq.com/blog/mqtt-security-fundamentals-tls-ssl ;

[HUG, 2009] : “Comparing Traditional Systems Analysis and Design with Agile Methodologies”, University of Missouri-St.Louis, 2009, Douglas Hughey, http://www.umsl.edu/~hugheyd/is6840/waterfall.html ;

[IWA] : “GATTOOL” , Nobuhiro Iwamatsu: http://sancho.ccd.uniroma2.it/cgi-bin/man/man2html?gatttool+1 ;

[JAM, 2017] : “Bluetooth Jamming : What For and How To ?”, Jammer-Store, 2017, https://www.jammer-store.com/bluetooth-jamming-what-for-and-how-to.html ;

[JAS1, 2016]: “Gattacker Framework Open-Source”, Github, 2016, Securing, Slawomir Jasek, https://github.com/securing/gattacker ;

[JAS2, 2016] : “Gattacker Presentation”, Black-Hat, 2016, Slawomir Jasek;

[JAS3, 2016] : “Gattacking Bluetooth Smart Devices”, Whitepaper, 2016, Slawomir Jasek ;

[KAS, 2016] : “10 Real World Applications of the IoT”, Analytics Vidhya, 2016, Swati Kashyap, https://www.analyticsvidhya.com/blog/2016/08/10-youtube-videos-explaining-the-real-world-applications-of-internet-of-things-IoT/ ;

[KAL] : “Kali Linux Tools Listing”, Kali Tools Main Page : https://tools.kali.org/tools-listing ;

[KHA et al. , 2012] : “Future Internet: The Internet Of Things Architecture, Possible Applications and Key Challenges”, ResearchGate, 2012, Rafiullah Khan, Sarmad Ullah Khan et al.;

[KLI et al., 2006] : “Introduction: Why Cross-Layer ? Its advantages and disadvantages”, Cross Layer Designs in WLAN Systems, 2006, Dzmitry Kliazovich et al. ;

[KOM et al., 2016] : “Incorporating Security Best Practices into Agile Teams” , ThoughtWorks, 2016, Chelsea Komlo et al. , https://www.thoughtworks.com/insights/blog/incorporating-security-best-practices-agile-teams ;

[KUM, 2017] : “Unpatchable Flaw in Modern Cars Allows Hacker sto Disable Safety Features”, The Hacker News, 2017, Mohit Kumar, https://thehackernews.com/2017/08/car-safety-hacking.html ;

[KWO et al. , 2016] : “Bluetooth Low Energy Security Vulnerability and Improvement Method”, IEEE International Conference on Consumer Electronics-Asia, 2016, Giwon Kwon et al. ;

[LIN, 2017] : “Automating Your Security Acceptance Tests” , OpenCredo , 2017 , Carlos Lindo, https://opencredo.com/automating-your-security-acceptance-tests/ ;

[LON, 2017] : “La sicurezza dentro l’internet degli oggetti”, Nova il Sole 24 Ore, 2017, Alessandro longo, http://nova.ilsole24ore.com/esperienze/la-sicurezza-dentro-linternet-degli-oggetti/?refresh_ce=1;

[MAR, 2008] : “Clean Code” , Prentice Hall, 2008, Robert Cencil Martin ;

[MOR et al., 2016]: “Identity and Access Management for the Internet Of Things – Summary Guidance”, Cloud Security Alliance, 2016, Arlene Mordeno, Brian Russell, et al., https://downloads.cloudsecurityalliance.org/assets/research/internet-of-things/identity-and-access-management-for-the-iot.pdf ;

[MQT] : “MQTT Project Home Page” : http://mqtt.org/ ;

[NGU, 2015] : “Integrating Security into Agile Software Development Methods” , University of Missouri- St.Louis, 2015, Tran Nguyen ;

[NRF]: “NRFConnect Android Application”, Github, Nordic Semiconductor, https://github.com/NordicSemiconductor/Android-nRF-Connect ;

[OWA,2001] : “Owasp Main_Page”, 2001, https://www.owasp.org/index.php/Main_Page;

[OWA, 2014] : “Internet of Things Top Ten”, Owasp IoT Project, 2014, https://www.owasp.org/images/7/71/Internet_of_Things_Top_Ten_2014-OWASP.pdf;

[OWA1] : “Owasp Internet of Things Project Main_Page” , https://www.owasp.org/index.php?title=OWASP_Internet_of_Things_Project;

[OWA2] : “Agile Software Development: Don’t Forget EVIL User Stories”, Owasp Web Project, https://www.owasp.org/index.php/Agile_Software_Development:_Don%27t_Forget_EVIL_User_Stories ;

[PAD et al. , 2017] : “Guide to Bluetooth Security”, NIST Special Pubblication, 2017, John Padgette et al. ;

[PEE, 2008] : “Agile Security Requirements Engineering” , Researchgate, 2008, Johan Peeters ;

[POZ, 2015] : “MQTT: il protocollo che rende possibile l’IoT”, SMAU Slideshare, 2015, Davide Pozza, https://www.slideshare.net/omnys_keynotes/mqtt-il-protocollo-che-rende-possibile-linternet-of-things ;

[REE, 2017] : “IoT Needs Embedded Cryptography” , EETimes Blog, 2017, Lynnette Reese, https://www.eetimes.com/author.asp?doc_id=1331566 ;

[REN, 2017] : “Bluetooth Pairing Part 4: LE Secure Connections – Numeric Comparison”, Bluetooth Blog, 2017, Kai Ren, https://blog.bluetooth.com/bluetooth-pairing-part-4 ;

[RIG, 2016] : “Negli Usa, 19 mld per la cyber security. E in Italia?”, Forumpa, 2016, Andrea Rigoni, http://www.forumpa.it/sicurezza/sicurezza-rigoni-negli-stati-uniti-19-miliardi-per-la-cybersecurity-e-noi ;

[ROU, 2011]: “pentest(penetration testing)”, TechTarget, 2011, Margaret Rouse, http://searchsoftwarequality.techtarget.com/definition/penetration-testing ;

[ROU, 2016] : “Internet Of Things (IoT)” , Techtarget IoT Agenda, 2016, Margaret Rouse, http://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT ;

[ROU, 2017] : “Test-driven development (TDD)”, TechTarget, 2017, Margaret Rouse, http://searchsoftwarequality.techtarget.com/definition/test-driven-development;

[RYA, 2013] : “Crackle open-source project”, Github, 2013, Mike Ryan : https://github.com/mikeryan/crackle ;

[SAL, 2015] : “Networking Protocols and Standards for Internet Of Things”, Washington University in St.Louis: Computer Science & Engineering, 2015, Tara Salman, https://www.cse.wustl.edu/~jain/cse570-15/ftp/iot_prot/ ;

[SAW, 2008] : “Internal vs. External Penetration Testing”, DARKReading, 2008, John H. Sawyer , https://www.darkreading.com/risk/internal-vs-external-penetration-testing/d/d-id/1129881? ;

[SCA, 2010] : “Continuous Delivery Pipeline” , Scaled Agile Framework, 2010, http://www.scaledagileframework.com/continuous-delivery-pipeline/ ;

[SHA, 2017] : “Artificial Intelligence and machine learning offer new possibilities for improving IoT security”, Zdnet, 2017, Mary Shacklett, http://www.zdnet.com/article/artificial-intelligence-and-machine-learning-offer-new-possibilities-for-improving-iot-security/;

[SHA et al. , 2017] : “Summary of an Open Discussion on IoT and Lightweight Cryptography”, Early Symmetric Crypto workshop, 2017, Adi Shamir et al. ;

[SIG] : “Bluetooth SIG Proprietary Information Security, Bluetooth Low Energy Security”, Bluetooth, 2011;

[SIL, 2015] : “Gartner prevede 20 miliardi di oggetti IoT connessi nel 2020”, Silicon, 2015, http://www.silicon.it/networks/gartner-prevede-20-miliardi-di-oggetti-IoT-connessi-nel-2020-88394?inf_by=59634d2c671db8f5738b4651;

[SIN et al., 2015] : “Secure layers based architecture for the Internet Of Things”, 2015 IEEE 2nd World Forum, 2015, Dhananjay Singh, Gaurav Tripathi, Antonio Jara;

[SON] : “SonarQube” : https://www.sonarqube.org/ ;

[SOU, 2017] : “IoT Threat Scenarios: Protection for IoT Security & Privacy”, Entrepreneur Network, 2017, Dinesh Soundararajan, https://www.entrepreneur.com/article/292104 ;

[SPI, 2007] : “BlueSniff: Eve meets Alice and Bluetooth”, ResearchGate, 2007, Dominic Spill, Andrea Bittau ;

[STE et al., 2014] : “Learning Agile: Understaing Scrum, XP, Lean and Kanban”, O’Reilly Media, 2014, Andrew Stellman, Jennifer Greene ;

[STR] : “STRIDE security” : https://en.wikipedia.org/wiki/STRIDE_(security);

[THO, 2014] : “Abuser Story – User Storie sto Prevent Hacking” , It’s a delivery Thing, 2014, Steven Thomas, http://itsadeliverything.com/abuser-story-user-stories-to-prevent-hacking ;

[TOW, 2014] : “Introduction To Bluetooth Low Energy” , Adafruit, 2014, Kevin Townsend, https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt# ;

[TRI, 2014] : “Bluetooth Low Energy Security”, Teledyne Lecroy, 2014, John Trinkle, http://www.fte.com/webhelp/bpa600/Content/Documentation/WhitePapers/BTLE/BluetoothSmartSecurity.htm ;

[UNU et al. , 2017] : “IT threat evolution Q2 2017. Statistics”, Securelist , 2017, Roman Unuchek, Fedor Sinitsyn et al. https://securelist.com/it-threat-evolution-q2-2017-statistics/79432/ ;

[VER, 2017] : “Owasp Top 10 Vulnerabilities”, VeraCode , 2017 https://www.veracode.com/directory/owasp-top-10 ;

[WIK] : “One Way Function”, Wikipedia: https://en.wikipedia.org/wiki/One-way_function ;

[WIK1]: “WhiteHat definition”: https://it.wikipedia.org/wiki/White_hat ;

[WIR] : “Wireshark Home Page” : https://www.wireshark.org/ ;

[WOL, 2012] : “What is HMAC Authentication and why is it useful?” , Mark Wolfe’s Blog, 2012, Mark Wolfe, https://www.wolfe.id.au/2012/10/20/what-is-hmac-authentication-and-why-is-it-useful/ ;

[ZUL et al. , 2011] : “Security Backlog in Scrum Security Process” , 5th Malaysian conference in Software Engineering (MySEC), 2011, Zulkarnain Azham et al. ;

Ringraziamenti

Ho realizzato il lungo progetto di ricerca raccontato in questo lavoro durante gli ultimi sei mesi trascorsi all'interno dell'azienda Gruppo Scai di Bologna, e comincio dal ringraziare la mia relatrice Prof.ssa Montanari che mi ha condotto ed illuminato all'interno di un ambito molto importante per quanto complesso, cercando di tirare fuori il meglio di me ed invogliando la mia curiosità e passione al fine di raggiungere ottimi risultati. In particolare ringrazio il mio Tutor aziendale Ciro, che mi ha sbloccato da molte situazioni critiche e che mi ha seguito con passione e fiducia per tutti questi mesi, e ringrazio inoltre il mio collega e caro amico Nicolò per i miei primi mesi di lavoro, sempre disponibile e di grande aiuto. Entrambi mi hanno aiutato nel realizzare un lavoro di massimo impegno, che con grande orgoglio rimarrà un riferimento all'interno dell'azienda e nella personale crescita professionale. Non di meno importanza sono stati per me gli altri ragazzi dell'azienda, che mi hanno subito accolto in un clima familiare, trattandomi da collega prima, e da amico poi: ringrazio quindi Enrico, Matteo, Dario, Valentina, Marco e Fabrizio.

È sorprendente cosa questo "semplice" percorso universitario mi abbia lasciato in tutti questi anni, in particolar modo negli ultimi due anni da fuorisede a Bologna. Oltre alla possibilità di avere le competenze per lavorare in un ambito che amo e che approfondisco sempre con forte passione, mi ha permesso di esplorare a fondo relazioni, luoghi ed esperienze in maniera veramente toccante. Questo periodo segnerà senz'altro tutta la mia vita: ciò che sarò dipenderà in gran parte da tutte le esperienze che ho vissuto e dalle persone che ho incontrato in questi anni.

Quindi voglio ringraziare innanzitutto chi mi ha permesso di fare questa esperienza di vita: i miei genitori Franco e Cinzia, che mi hanno permesso di studiare attraverso i loro sacrifici, la loro costante fiducia e ammirazione per la mia passione e per la voglia di portare a termine questo percorso. Avere l'occasione di trasmettere ai propri figli questo stesso sostegno per la loro passione sarà uno degli obiettivi della mia vita, a conferma del grande messaggio che hanno trasmesso i miei genitori verso di me.

Ringrazio di cuore i miei parenti più stretti, che non mi hanno mai fatto mancare il loro affetto anche da lontano, sempre disponibili e vicini a me: il mio grande fratello Andrea, la mia cara nonna Giovanna, i miei zii e le mie amate cugine.

Non posso citarli tutti, ma ringrazio immensamente i miei amici più stretti, consapevole che l'amicizia sia il sentimento più vero e sincero che si possa provare: sia quelli con cui ho condiviso la mia esperienza di vita a Bologna (la mia seconda famiglia: Michele, Riccardo e Andrea), che quelli che mi accolgono sempre quando torno nella mia amata città natale, Pesaro (tra gli altri: Tommaso, Giacomo, Fabio, Eugenio, Matteo, Davide, Simone, Alessandro, Mattia e Il Capitano).

Ringrazio di cuore la mia ragazza Giulia, la persona a cui forse tengo più, con cui ho vissuto l'esperienza della lontananza in questi anni da fuorisede, e che nonostante il distacco mi ha reso consapevole della solidità dei nostri sentimenti e del legame profondo che ci tiene uniti da tanti anni, e che spero ci leghi ancora per molto tempo.

Uno dei legami più forti che ho provato nella mia vita è senza dubbio quello che mi lega con la persona a cui è dedicato questo lavoro, ovvero mio nonno Claudio, che mi sostiene e mi protegge da lassù.

Devo tutto questo traguardo a lui e a mia nonna Giovanna che mi hanno cresciuto educandomi e orientando il mio carattere verso la perenne voglia di imparare e di essere curioso, di sacrificarmi per gli altri e per ciò che conta veramente nella vita. Mio nonno è sempre stato premuroso per il mio percorso di studi, contentissimo e fiero di me qualunque risultato raggiungessi: la forza di condurre questo percorso è stata gran parte sua.

E, dunque, anche questa Tesi.