Università di Genova | DIPARTIMENTO DI INFORMATICA, BIOINGEGNERIA, ROBOTICA E INGEGNERIA DEI SISTEMI

Scuola di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Anno Accademico 2022-2023

Binary Instrumentation per il monitoraggio runtime di sistemi Internet of Things tramite l'Impiego di Falco

Federico Elia

Relatore:

Prof. Davide Ancona

INDICE

Indice delle figure

CAPITOLO I

INTRODUZIONE

1.1 Obiettivi

La tesi si propone di esplorare l'universo dell'Internet of Things (IoT), delineando inizialmente una panoramica comprensiva di questo fenomeno tecnologico e della sua importanza nel contesto attuale. L'obiettivo primario è di offrire una definizione chiara dell'IoT e di illustrare la vasta gamma di applicazioni che questa tecnologia abilita in diversi settori, sottolineando come l'IoT stia rivoluzionando le modalità di interazione tra oggetti fisici e il mondo digitale. Particolare attenzione viene dedicata all'identificazione delle possibili falle di sicurezza nei sistemi IoT, esaminando il loro impatto sulla privacy e sulla sicurezza dei sistemi. Nel prosieguo, la tesi si addentra nello studio del protocollo MQTT, una tecnologia chiave nei sistemi IoT per la sua efficacia nella gestione della comunicazione tra dispositivi. Si fornisce una spiegazione dettagliata del funzionamento di base di MQTT e delle ragioni della sua importanza, mettendo in luce i problemi di sicurezza che lo affliggono e le possibili soluzioni per mitigarli.

Questo studio si prefigge di quantificare e comprendere gli effetti dell'overhead causato dell'instrumentazione binaria e il monitoraggio di dispositivi intelligenti, valutandone l'impatto sulle prestazioni di un sistema IoT reale. La sezione esplora quindi il background tecnico indispensabile per assimilare gli strumenti e le metodologie utilizzate nel progetto, con un'enfasi specifica sull'instrumentazione binaria. Si introduce il concetto di instrumentazione binaria, si analizzano le sue applicazioni nel monitoraggio e si descrivono gli strumenti e le tecnologie pertinenti, gettando le fondamenta per le future elaborazioni della tesi.

Un focus particolare è riservato a Falco, uno strumento emergente nel campo della sicurezza. Si offre una panoramica esaustiva su Falco, sottolineandone le principali funzionalità e ambiti di applicazione, con particolare attenzione all'integrazione di questa tecnologia per potenziare la sicurezza nei sistemi IoT. Tale integrazione mira a svelare come Falco possa essere efficace nel prevenire o attenuare le minacce alla sicurezza, offrendo una visione concreta di come l'instrumentazione binaria interagisca e influenzi sistemi IoT operativi.

1.2 Motivazioni

La tesi si incentra sull'esplorazione e l'innovazione nel campo dell'IoT, una tecnologia che sta rivoluzionando il modo in cui viviamo, lavoriamo e interagiamo con il mondo che ci circonda. L'IoT, con la sua vasta rete di dispositivi connessi, offre possibilità illimitate per migliorare l'efficienza, la comodità e la sicurezza delle nostre vite quotidiane. Tuttavia, con questa pervasività, emergono sfide significative, in particolare riguardo alla sicurezza e alla protezione della privacy degli utenti. La motivazione principale di questa tesi è quindi duplice: da un lato, vuole fornire una comprensione approfondita dell'IoT, evidenziando la sua importanza nel contesto attuale ed esplorando le sue applicazioni in vari settori. Dall'altro lato, mira a indagare e affrontare le vulnerabilità intrinseche dei sistemi IoT, concentrandosi in particolare sul protocollo MQTT, ampiamente utilizzato per la comunicazione tra dispositivi. Questo protocollo, nonostante la sua efficacia ed efficienza, presenta criticità in termini di sicurezza che possono essere sfruttate da malintenzionati.

1.3 IoT

L'Internet delle Cose (IoT) è un paradigma emergente che trasforma la nostra interazione con il mondo, permettendo ai dispositivi elettronici e ai sensori di comunicare attraverso la rete, semplificando e arricchendo le nostre vite. L'IoT utilizza dispositivi intelligenti e Internet per fornire soluzioni innovative a varie sfide e problemi relativi a vari settori aziendali, governativi e pubblico/privato in tutto il mondo. L'IoT sta progressivamente diventando un aspetto importante della nostra vita che può essere percepito ovunque intorno a noi.

Nel complesso, l'IoT è un'innovazione che mette insieme un'ampia varietà di sistemi intelligenti, framework e dispositivi e sensori intelligenti.

Smart city, case intelligenti, controllo dell'inquinamento, risparmio energetico, trasporti intelligenti, industrie intelligenti sono trasformazioni dovute all'IoT.

L'IoT è al centro di ricerche significative volte a potenziare le tecnologie esistenti. Nonostante i progressi, l'IoT presenta ancora sfide complesse che ne limitano il pieno potenziale. Queste problematiche spaziano dalle questioni tecniche legate alla sicurezza e alla privacy dei dati, fino agli aspetti sociali ed ambientali. Per massimizzare i benefici dell'IoT, è fondamentale affrontare tali sfide considerando vari aspetti come le applicazioni specifiche, le tecnologie coinvolte e l'impatto più ampio su società e ambiente. Ciò è possibile fornendo un maggiore processo decisionale

automatizzato in tempo reale e facilitando gli strumenti per ottimizzare tali decisioni. L'integrazione delle energie rinnovabili e l'ottimizzazione dell'uso dell'energia sono fattori chiave per le transizioni energetiche sostenibili e la mitigazione del cambiamento climatico, l'IoT può essere impiegato per migliorare l'efficienza energetica, aumentare la quota di energia rinnovabile e ridurre l'impatto ambientale dell'uso dell'energia (Ali, 2015).

1.4 Struttura della tesi

Questa tesi è articolata in tre parti principali, seguendo una struttura logica che guida il lettore dall'introduzione al contesto e alla rilevanza dell'IoT nel panorama tecnologico odierno, attraverso l'approfondimento di aspetti tecnici specifici, fino alla presentazione di un contributo originale di ricerca del contributo della tesi:

1. La prima parte introduce il contesto e la rilevanza dell'IoT nel panorama tecnologico attuale, evidenziando come la sua pervasività incida profondamente su molti aspetti della società moderna. Vengono discusse le potenzialità dell'IoT in vari ambiti, dall'automazione domestica alla smart city, dall'industria 4.0 all'assistenza sanitaria, mettendo in luce le incredibili opportunità offerte da questa tecnologia ma anche le sfide che ne derivano, soprattutto in termini di sicurezza. Il lavoro si concentra su MQTT, un protocollo chiave per la comunicazione nei sistemi IoT, descrivendo il suo funzionamento e il suo ruolo critico nell'ecosistema IoT. Vengono analizzate le problematiche di sicurezza legate a MQTT, evidenziando come la sua implementazione possa esporre i sistemi a rischi significativi e quali misure possano essere adottate per mitigare tali vulnerabilità.
2. Nella seconda parte, ci si addentra nel background tecnico necessario per comprendere gli strumenti utilizzati nel progetto, con un focus sull'instrumentazione binaria e su Falco.

   Viene fornita una panoramica dell'instrumentazione binaria, delle sue applicazioni nel monitoraggio e della sua importanza nella rilevazione e prevenzione di attacchi informatici. Successivamente, viene introdotto Falco, descrivendo le sue funzionalità chiave e il modo in cui può essere integrato nei sistemi IoT per rafforzarne la sicurezza.
3. Nella terza parte, la tesi si concentra sul contributo principale della tesi, inizialmente l'attenzione si focalizza sul dettagliato processo di instrumentazione binaria implementato mediante l'utilizzo di Falco, ponendo

particolare enfasi su come quest'ultimo venga configurato con regole personalizzate per monitorare specifici comportamenti o eventi all'interno dei sistemi IoT. Parallelamente, si introduce un Adapter che agisce come intermediario tra Falco e il monitor. Questo Adapter ha il compito cruciale di elaborare i dati ricevuti da Falco in modo che possano essere interpretati efficacemente dal sistema di monitoraggio sviluppato in Prolog ed in RML (Runtime Monitoring Language). L'uso di RML (Runtime Monitoring Language) è cruciale in questo contesto, poiché viene utilizzato per definire le specifiche del monitor in maniera semplice e coincisa.

Questa analisi è stata focalizzata sulla valutazione dell'efficacia degli strumenti proposti per la mitigazione dei rischi di sicurezza nei sistemi IoT. Particolare attenzione è stata dedicata alla misurazione dell'overhead generato dall'implementazione di tali strumenti, sia individualmente sia in combinazione. L'obiettivo è stato quello di offrire una panoramica dettagliata sulle implicazioni dell'overhead associato a ciascuno strumento, permettendo così di trovare un equilibrio ottimale tra sicurezza e prestazioni.

Riassumendo questa tesi mira a valutare l'efficacia di tali strumenti e metodologie nell'ottimizzare la sicurezza dei sistemi IoT, contribuendo così alla loro maggiore resilienza contro le minacce informatiche.

CAPITOLO II

INTERNET OF THINGS (IOT): PANORAMICA

2.1 IOT: Definizione e importanza dell'IoT nel contesto attuale

L'IoT costituisce una delle più significative rivoluzioni tecnologiche del nostro tempo. Questo concetto si riferisce a un'infrastruttura globale che collega fisicamente il mondo digitale ai dispositivi e agli oggetti del mondo reale. Gli elementi che compongono questa rete - che possono variare da semplici sensori e attuatori domestici a complessi sistemi industriali - sono equipaggiati con la tecnologia necessaria per raccogliere, inviare e ricevere dati attraverso la rete.

Questa inter-connettività permette ai dispositivi di interagire non solo tra loro ma anche con altre piattaforme e servizi, facilitando un'ampia gamma di applicazioni. Ad esempio, in un ambiente domestico, l'IoT può essere utilizzato per gestire la temperatura, l'illuminazione e la sicurezza in modo automatico e remoto. Nel settore industriale, può migliorare l'efficienza dei processi produttivi attraverso la raccolta e l'analisi in tempo reale dei dati operativi.

I dispositivi IoT sono dotati di sensori che rilevano vari tipi di dati ambientali o operativi, come temperatura, umidità, movimento o livelli di luminosità. Questi dati vengono poi trasmessi a sistemi in grado di elaborarli, spesso utilizzando tecnologie cloud per l'analisi e la persistenza di voluminose serie di dati. Le informazioni raccolte ed elaborate permettono al sistema di prendere decisioni autonome, eventualmente mediate dall'intervento umano, per operare sugli attuatori e perseguire al meglio gli obiettivi prefissati.

L'IoT sta rivoluzionando interi settori: nell'industria abilita il monitoraggio in tempo reale della produzione, l'ottimizzazione della supply chain e il miglioramento dell'efficienza; in sanità permette il monitoraggio da remoto di pazienti tramite dispositivi indossabili; nelle case sta alla base della domotica e della gestione intelligente di elettrodomestici e sistemi (Ali, 2015).

Nonostante la diversità della ricerca sull'IoT, la sua definizione rimane alquanto confusa. Esistono diverse definizioni che riflettono prospettive diverse; di seguito vengono riportate quelle più importanti in letteratura:

- "L'IoT è la rete di oggetti fisici che contengono tecnologia incorporata per comunicare e percepire o interagire con i loro stati interni o l'ambiente esterno.

L'IoT comprende un ecosistema che include cose, comunicazione, applicazioni e analisi dei dati" (Vuppalapati, 2019).

- "L'IoT è un gruppo di infrastrutture che interconnettono sensori e/o attuatori con capacità di calcolo (limitate) e ne consentono la gestione, l'accesso e il trasferimento dei dati che generano su Internet senza la necessità di risorse umane l'interazione tra uomo o uomo-computer" (Dorsemaine et al., 2015).
- "L'IoT si riferisce a un sistema decentralizzato di dispositivi potenziati con capacità di rilevamento, elaborazione e rete" (Kortuem, 2009).

Nell'industria, l'IoT gioca un ruolo cruciale nell'automazione e nella digitalizzazione dei processi, consentendo un'indagine dettagliata e in tempo reale delle operazioni, una gestione ottimizzata della catena di approvvigionamento e un incremento dell'efficienza energetica e produttiva.

Nel settore sanitario, dispositivi indossabili e altri strumenti IoT forniscono dati vitali per il monitoraggio da remoto dei pazienti, migliorando la qualità delle cure e la rapidità di intervento.

Nelle abitazioni, l'IoT trasforma gli spazi in ambienti intelligenti, in cui ogni dispositivo è in grado di comunicare per migliorare comfort e sicurezza. Tuttavia, l'enorme potenziale dell'IoT è accompagnato da sfide significative, accentuate dalla limitata potenza di calcolo dei dispositivi IoT, che spesso non possono supportare protocolli di sicurezza avanzati o ospitare software antivirus complessi. Di conseguenza, diventano obiettivi vulnerabili per gli attacchi informatici, esponendo i sistemi a rischi significativi (Ali, 2015).

2.1.1 Panoramica delle applicazioni IoT in vari settori

Le reti IoT generano un'enorme quantità di dati aggregati tramite degli oggetti intelligenti. Ogni due anni, le dimensioni dei dati raddoppiano e si prevede che raggiungano i 163 Zettabyte nel 2025 (Beecks, 2018).

L'IoT e il cloud computing servono entrambi ad aumentare l'efficienza per le aziende e l'industria. L'IoT genera un'enorme quantità di dati, con il cloud computing che fornisce spazio di archiviazione e potenza di calcolo per l'analisi dei dati. L'integrazione dell'IoT con le tecnologie mediche consente il sistema di monitoraggio in tempo reale e l'accesso ai dati per migliorare la salute dei pazienti (Mubeen, 2018).

Ad esempio, i dispositivi IoT possono essere utilizzati per tracciare la posizione in tempo reale di apparecchiature mediche come sedie a rotelle, pompe di ossigeno e altre

apparecchiature di monitoraggio. I sistemi IoT basati sulla tecnologia wireless vengono utilizzati per monitorare un'ampia gamma di sensori sul campo in grado di rilevare e misurare vari fenomeni fisici come attività vulcanica, inondazioni e incendi (Amarlingam, 2016).

Questi esempi sono solo alcune delle tipologie di applicazioni sviluppate nel campo dell'IoT. La Figura 1 illustra alcune delle aree che beneficiano della tecnologia IoT (Amarlingam, 2016).

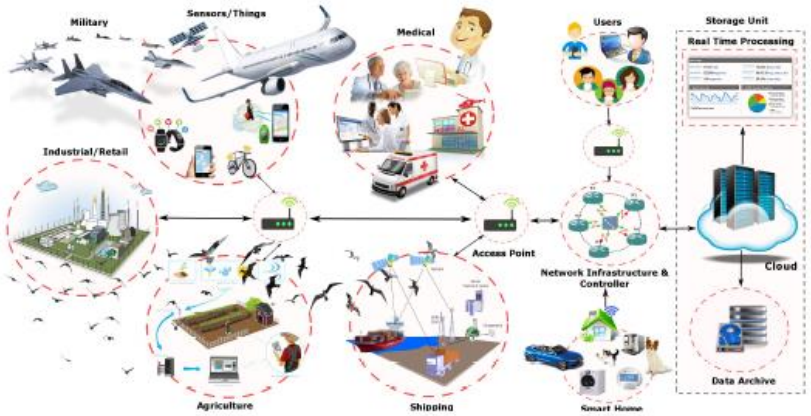

Figura 1. Alcuni domini applicativi IoT

Collegando miliardi di dispositivi ad internet, l'IoT ha creato un'infinità di applicazioni che toccano ogni aspetto della vita umana (figura 2); per citarne solo alcuni:

- **Smart Cities -** l'utilizzo di sensori, strutture di rete e sistemi di integrazione basati su cloud per fornire ai cittadini servizi e infrastrutture e dare loro accesso a una vasta gamma di informazioni in tempo reale sull'ambiente urbano, su cui basare decisioni e azioni (Jin et al., 2014).
- **Smart Grids -** reti elettriche in grado di integrare i comportamenti e le azioni di tutti gli utenti ad essa collegati in modo efficiente in termini di costi, al fine di garantire un sistema elettrico economicamente efficiente e sostenibile con basse perdite ed elevati livelli di qualità e sicurezza dell'approvvigionamento e della sicurezza (Guillemin, 2014).
- **Connected Health -** la raccolta, l'aggregazione e l'efficace elaborazione e di informazioni indicative della salute fisica e mentale (Hassanalieragh, 2015).
- **Smart Homes -** una comoda configurazione domestica in cui elettrodomestici e dispositivi possono essere controllati automaticamente o

in remoto da qualsiasi luogo connesso a Internet utilizzando un dispositivo mobile o un altro dispositivo in rete (Ghayvat, 2015).

- **Connected Cars** - automobili in grado di connettersi a Internet e condividere vari tipi di dati (posizione, velocità, stato di parti dell'auto, ecc.) con applicazioni back-end. utili, per esempio, per la gestione di attività di car sharing o di sistemi di allerta in caso di incidenti o problemi di viabilità (Dhall, 2017).

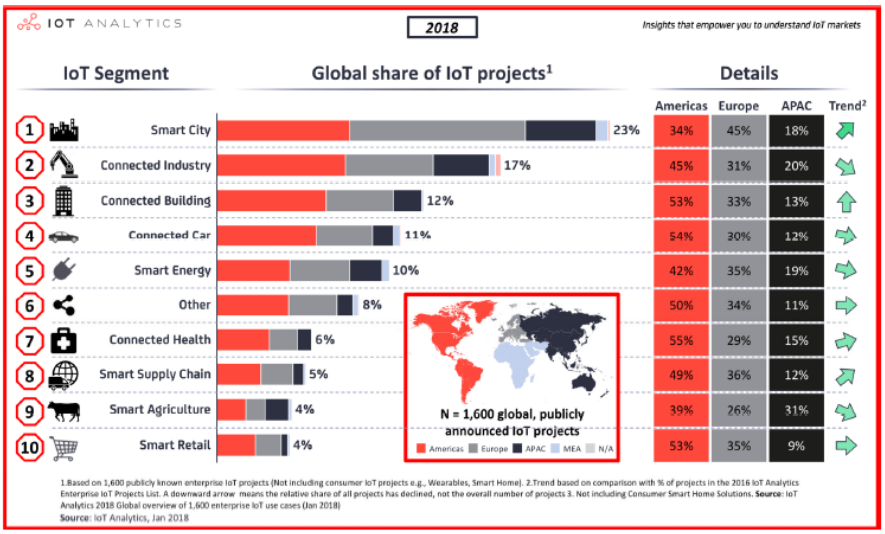

Figura 2. Le applicazioni IoT più popolari

Le applicazioni IoT possono essere suddivise in base ai loro requisiti principali in tre categorie (Dıaz, 2016):

1. **In tempo reale:** applicazioni che richiedono il monitoraggio dei dati e il supporto decisionale in tempo reale, come accade, per esempio, per i domini Connected Health, Smart Farming e Smart Supply Chain;
2. **Analisi dei dati:** applicazioni focalizzate sull'analisi dei dati, ad esempio, di tipo Smart Retail, Smart City e Smart Grid, che si affidano all'analisi dei dati per ottimizzare rispettivamente il business, le città e le reti elettriche;
3. **Interazione con i dispositivi:** le applicazioni si concentrano sulle relazioni tra i dispositivi. In Smart Home, Wearables e Industrial Internet, l'interazione con i dispositivi è un obiettivo chiave.

2.1.2 Possibili falle sicurezza nei sistemi IoT e il loro impatto

L’aspetto della sicurezza di questa tecnologia è significativo poiché recenti indagini e tendenze hanno evidenziato la sua importanza in questo settore. Secondo il National

Institute of Standards and Technology, la garanzia delle informazioni è definita come "misure che salvaguardano e preservano le informazioni e i sistemi informativi garantendone la segretezza, la verifica, l'integrità, la disponibilità e il non ripudio". Questi passaggi includono "la fornitura del ripristino delle reti di informazione integrando capacità di sicurezza, rilevamento e risposta". Poiché i sistemi IoT coinvolgono dispositivi fisici, reti di comunicazione e risorse di dati, questi cinque pilastri della garanzia delle informazioni sono rilevanti come requisiti di sicurezza (Russell, 2016).

I requisiti di sicurezza dell'IoT sono fondamentali per garantire il funzionamento sicuro dei dispositivi interconnessi e dei dati da essi prodotti. Forti meccanismi di autenticazione e controllo degli accessi per prevenire l'accesso non autorizzato e difendersi dagli attacchi informatici sono requisiti essenziali di sicurezza dell'IoT. Questi meccanismi devono essere in grado di identificare e autenticare utenti e dispositivi, controllare l'accesso ai dati sensibili e fornire autorizzazioni granulari per garantire che solo le entità autorizzate possano accedere al sistema. Inoltre, i dati generati dai dispositivi IoT dovrebbero essere crittografati e protetti per garantire privacy e riservatezza ed essere protetti da manomissioni per garantirne l'integrità e l'autenticità. La sicurezza della rete e dei dispositivi è un altro aspetto vitale della sicurezza IoT. I dispositivi e i sistemi IoT devono essere protetti dagli attacchi basati sulla rete. Anche gli attacchi fisici, come la distruzione, il furto e la manomissione, dovrebbero essere prevenuti dai meccanismi di sicurezza integrati nei dispositivi IoT (Aldweesh, 2020).

L'IoT combina il mondo fisico e quello connesso a Internet per fornire una collaborazione intelligente tra le entità fisiche e i loro ambienti circostanti. In genere, i dispositivi IoT funzionano in una varietà di ambienti per raggiungere una varietà di obiettivi. Le aziende che operano in questo settore devono adottare misure di sicurezza estremamente rigorose, sia dal punto di vista informatico che fisico, per proteggere i propri dispositivi e le reti da potenziali minacce.

La complessità deriva non solo dalla varietà e dall'interdisciplinarità dei componenti coinvolti - che includono hardware, software, reti di comunicazione e algoritmi di elaborazione dati - ma anche dalla vastità e diversità degli scenari applicativi dell'IoT. Questa complessità aumenta esponenzialmente le superfici di attacco disponibili agli aggressori, rendendo più arduo il compito di mantenere i sistemi sicuri. Gli attacchi possono avvenire su molteplici fronti: dall'intercettazione delle comunicazioni

wireless, che sono spesso il mezzo attraverso cui i dispositivi IoT scambiano dati, al tentativo di accesso fisico diretto ai dispositivi, reso possibile dalla loro diffusione in ambienti aperti o facilmente accessibili al pubblico. Per affrontare efficacemente queste sfide, è necessario adottare un approccio alla sicurezza che sia onnicomprensivo e adattabile alle specifiche esigenze dell'IoT. Questo significa implementare soluzioni che non solo proteggano le comunicazioni e i dati, ma che siano anche progettate tenendo conto delle limitazioni intrinseche dei dispositivi IoT, come la limitata capacità di elaborazione e la scarsa autonomia energetica. Tali soluzioni dovrebbero includere l'uso di algoritmi di crittografia leggeri, sistemi di autenticazione robusti, protocolli di comunicazione sicuri e pratiche di sicurezza fisica (Aldweesh, 2020).

Di conseguenza, preservare la privacy o la sicurezza dei dispositivi basati sull'IoT è un compito complesso e difficile che ha suscitato un notevole interesse sia in ambito accademico che industriale. Dato che l'obiettivo principale di un sistema basato sull'IoT è fornire un accesso semplice a chiunque, ovunque e in qualsiasi momento, le possibilità di attacco aumentano. I dispositivi IoT generano grandi quantità di dati, che vengono trasmessi sulle reti, rendendoli vulnerabili alle minacce informatiche. Di conseguenza, proteggere la rete e i dati nell'IoT è essenziale per la sicurezza e la protezione dell'intero sistema.

Le misure di sicurezza della rete forniscono le basi per proteggere i dati in transito, mentre le misure di sicurezza dei dati salvaguardano i dati in transito e a riposo. Per garantire il funzionamento sicuro e protetto dei dispositivi e dei sistemi IoT, è necessario adottare una strategia di sicurezza completa che includa misure di sicurezza sia della rete che dei dati (Chaabouni, 2019).

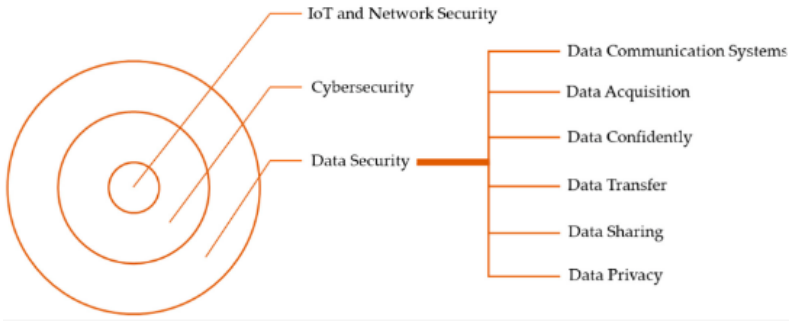

Figura 3. Sicurezza delle informazioni IoT: quanto è importante?

Le difficoltà in quest'area sono legate a tre fattori interconnessi: innanzitutto, a causa della natura limitata delle risorse e della mobilità dei sistemi IoT, i metodi di sicurezza dei dati devono operare in modo da consentire un consumo di risorse molto limitato.

In secondo luogo, numerose strutture IoT sono supportate dalla condivisione dei dati; tuttavia, in contesti sensibili ai dati, la segretezza è della massima importanza, il che spesso presenta numerosi problemi. In terzo luogo, la necessità di sicurezza dei dati aumenta notevolmente, in particolare nel caso di servizi o app IoT sensibili. La sicurezza IoT è una tattica di protezione e un meccanismo di difesa che protegge dagli attacchi che prendono di mira in particolare i dispositivi IoT connessi fisicamente. La sicurezza della rete protegge la rete e i dati in essa contenuti da intrusioni, assalti e altre minacce. Si tratta di un termine ampio e inclusivo che copre sia soluzioni software che hardware, nonché procedure, linee guida e configurazioni per l'uso della rete, l'accessibilità e l'elusione delle minacce in generale. I metodi di crittografia sono solo un aspetto del tema della protezione dei dati. Dalla combinazione dell'IoT con il paradigma dell'Industria 4.0 derivano numerosi vantaggi, tra cui un migliore sfruttamento dei dati dell'IoT. Ciò riguarda la condivisione delle informazioni e altre operazioni dipendenti dai dati che potrebbero aver luogo ovunque nel sistema, anche al di fuori dei confini dell'organizzazione. Sebbene i metodi di crittografia consentano uno scambio preferenziale di dati, questa parte elabora altre strategie per mantenere la riservatezza dei dati IoT (Figura 3). Esistono connessioni significative tra l'IoT e la sicurezza della rete, la sicurezza informatica e la sicurezza dei dati.

La protezione dei dispositivi e delle reti IoT è fondamentale per prevenire gli attacchi informatici. La sicurezza della rete protegge le reti che collegano i dispositivi IoT. La sicurezza informatica richiede la difesa dell'intero ecosistema IoT dagli attacchi informatici, inclusi dispositivi, reti e app (Taherdoost, 2023).

La sicurezza dei dati è la salvaguardia dei dati raccolti e comunicati dai dispositivi IoT. Ciò comporta la crittografia dei dati durante la trasmissione e la loro archiviazione in modo sicuro. Inoltre, le restrizioni di accesso e i sistemi di autenticazione sono fondamentali per prevenire l'accesso indesiderato ai dati sensibili. Queste misure di sicurezza dei dati sono essenziali per proteggere i dati raccolti e trasmessi dai dispositivi IoT. I dispositivi IoT sono suscettibili alle minacce informatiche e proteggerli implica l'installazione di misure di sicurezza della rete, sicurezza informatica e protezione dei dati (Al-Fuqaha, 2015).

2.2 MQTT: introduzione

MQTT (Message Queuing Telemetry Transport) è un protocollo di messaggistica leggero ed efficiente, ampiamente adottato nell'ambito dell'Internet delle Cose (IoT) per le sue eccellenti performance e per il suo modello di comunicazione basato su

publish/subscribe. Questo modello si rivela particolarmente adatto alle esigenze delle architetture IoT, in quanto permette ai dispositivi di inviare (publish) informazioni senza dover stabilire una connessione diretta con i riceventi (subscribers), facilitando così la scalabilità e l'efficienza della rete. I dispositivi IoT possono così comunicare tra loro e con i server in modo asincrono, ottimizzando l'uso della larghezza di banda e riducendo il consumo energetico, aspetti cruciali per dispositivi spesso caratterizzati da risorse limitate (La Marra, 2017).

2.2.1 Funzionamento base di MQTT

Il funzionamento base di MQTT si articola intorno a due entità principali: il client e il broker:

- Il client MQTT rappresenta un elemento essenziale all'interno dell'ecosistema MQTT, consentendo ai dispositivi di interagire con un broker centrale per lo scambio di messaggi.

  In qualità di publisher, il client MQTT trasmette informazioni legate a specifici topic.
- Quando funge da subscriber, si abbona ai topic di suo interesse per accogliere i messaggi che vi vengono diffusi. Questa architettura di messaggistica offre la flessibilità per cui un singolo client possa simultaneamente ricevere informazioni come subscriber e trasmetterne come publisher, interagendo con lo stesso broker.

  Inoltre, un client può essere iscritto a più topic, permettendogli di ricevere dati da molteplici fonti e di segmentare efficacemente le informazioni in base ai propri interessi o necessità operative.
- Il broker MQTT funge da cuore pulsante della rete MQTT, avendo il compito cruciale di gestire i flussi di messaggi. Riceve le comunicazioni dai publisher, le classifica in base ai topic e le distribuisce ai subscriber pertinenti, questo processo è fondamentale per garantire che l'informazione venga distribuita correttamente all'interno della rete, mantenendo l'efficienza e l'efficacia della comunicazione. In ambienti basati su Linux, ci sono diversi broker MQTT disponibili, molti dei quali open source, come Mosquitto, EMQX e HiveMQ.

Il vantaggio del sistema di pubblicazione/sottoscrizione è che publisher e subscriber non si conoscono perché esiste il broker che fa da intermediario, questo crea un disaccoppiamento temporale che rende impossibile la connessione simultanea del

subscriber e del publisher, in modo che il cliente rimanga a ricevere i dati ritardati in precedenza (Atmoko, 2017).

Di solito, nell'architettura MQTT, diversi sensori pubblicano periodicamente i risultati delle loro misurazioni su un determinato topic. Ogni dispositivo registrato a un topic specifico riceverà un messaggio dal broker ogni volta che il topic viene aggiornato, la figura 4 mostra l'esempio dell'utilizzo del protocollo MQTT.

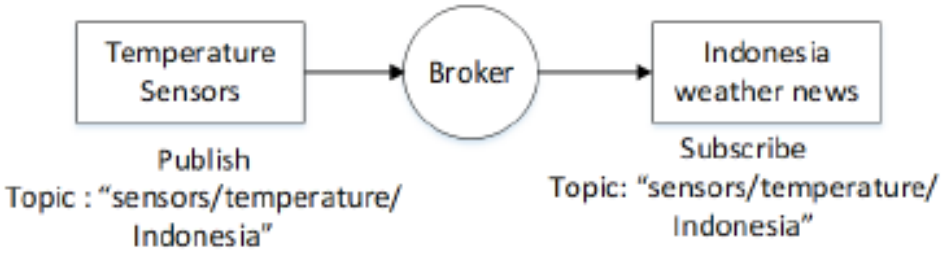

Figura 4. Mostra l'esempio dell'utilizzo del protocollo MQTT

I topic nei sistemi di messaggistica basati su MQTT seguono una struttura gerarchica simile ai path nei file system. Questo significa che i messaggi possono essere pubblicati e sottoscritti utilizzando topic che indicano livelli di categorizzazione tramite separatori (solitamente il carattere '/'). Per esempio, un topic potrebbe avere la struttura casa/soggiorno/temperatura per indicare la temperatura nel soggiorno di una casa. Le wildcard sono caratteri speciali usati nelle sottoscrizioni ai topic per indicare uno o più livelli di questa gerarchia, permettendo così sottoscrizioni più flessibili. Le due wildcard comunemente usate sono (Hwang, 2022):

- \+ sostituisce un singolo livello del topic. Per esempio, casa/+/temp corrisponde sia a casa/soggiorno/temp che a casa/cucina/temp.
- # sostituisce zero o più livelli alla fine del topic. Per esempio, casa/# corrisponde a tutti i messaggi che iniziano con casa/.

L'uso delle wildcard, tuttavia, può avere implicazioni per la sicurezza. Ecco alcune considerazioni:

- **Sovra-sottoscrizione:** L'uso improprio delle wildcard potrebbe portare un client a ricevere un volume eccessivo di messaggi, inclusi quelli non necessari o indesiderati. Questo può esporre il client a informazioni sensibili non destinate a lui o sovraccaricare il client e la rete.

- **Filtro inefficace:** Una sottoscrizione troppo generica potrebbe non filtrare efficacemente i messaggi, portando alla ricezione di dati irrilevanti o potenzialmente pericolosi.
- **Accesso indesiderato:** Se le politiche di sicurezza non fossero ben configurate, un client potrebbe utilizzare le wildcard per accedere a topic che dovrebbero essere riservati o protetti, ottenendo così accesso a informazioni sensibili.

MQTT fornisce tre livelli di Qualità del Servizio (QoS) per garantire l'affidabilità della consegna dei messaggi tra client e broker. Ecco un riassunto dei tre livelli (Atmoko, 2017):

- Al massimo una volta (QoS 0): Il messaggio viene inviato una sola volta senza garanzie che il destinatario lo riceva.
- Almeno una volta (QoS 1): Garantisce che il messaggio venga consegnato al destinatario almeno una volta.
- Esattamente una volta (QoS 2): È il livello più sicuro e garantisce che ogni messaggio venga ricevuto esattamente una volta dal destinatario.

2.2.2 Importanza di MQTT nei sistemi IoT

La sua importanza nei sistemi IoT è profondamente radicata nella sua capacità di facilitare una comunicazione affidabile ed efficiente tra dispositivi, spesso chiamati "things", e il server o broker MQTT che coordina la distribuzione dei messaggi. I dispositivi IoT, come sensori o attuatori, pubblicano messaggi su argomenti specifici, che sono gestiti da un broker MQTT. Altri dispositivi o servizi possono sottoscrivere uno o più di questi argomenti per ricevere i messaggi pertinenti. Questa modalità di funzionamento migliora la scalabilità del sistema e facilitando l'integrazione di nuovi dispositivi. La sicurezza è un altro aspetto fondamentale affrontato da MQTT, che offre meccanismi come la crittografia SSL/TLS per proteggere i dati trasmessi tra i dispositivi e il broker. Questo è particolarmente importante nell'era dell'IoT, dove la sicurezza dei dati e la protezione dalla manipolazione o dall'accesso non autorizzato sono critiche. L'efficienza di MQTT nel gestire la comunicazione tra una vasta gamma di dispositivi lo rende ideale per applicazioni IoT diverse, che vanno dalla domotica al monitoraggio industriale, dall'agricoltura intelligente alle soluzioni di smart city. Per esempio, in un'applicazione di monitoraggio ambientale, sensori dislocati in varie posizioni possono pubblicare periodicamente dati su temperatura, umidità o qualità dell'aria su argomenti specifici gestiti da un broker MQTT. Un'applicazione di analisi

centralizzata può sottoscrivere questi argomenti per raccogliere e processare i dati, generando allarmi o azioni automatiche in base ai dati ricevuti.
Il protocollo MQTT è progettato per funzionare sopra TCP/IP, offrendo un metodo efficiente e leggero per la trasmissione di messaggi. Una delle caratteristiche distintive di MQTT è la sua minima dimensione del pacchetto dati, con un overhead che inizia da appena 2 byte. Questa ridotta dimensione del pacchetto si traduce in un minor consumo di banda e, di conseguenza, in un'ottimizzazione del consumo energetico, consentendo loro di rimanere in modalità "sleep" per periodi prolungati. Questo approccio è cruciale per gli scenari IoT, dove i dispositivi possono essere alimentati a batteria e dislocati in ambienti remoti o difficilmente accessibili, rendendo essenziale la loro capacità di operare per lunghi periodi senza manutenzione (Atmoko, 2017) (Mishra, 2020).

2.2.3 Problemi di sicurezza del protocollo MQTT

La sicurezza in MQTT può essere compromessa a causa della mancata implementazione di misure di sicurezza robuste, lasciando i dispositivi e i dati vulnerabili a vari tipi di attacchi. Uno dei principali problemi di sicurezza con MQTT è l'autenticazione insufficiente. Di default, MQTT non richiede autenticazione forte, permettendo a chiunque conosca l'indirizzo del broker di pubblicare o sottoscrivere ai topic, esponendo i sistemi a rischi di intercettazioni o manipolazioni indebite dei messaggi. Questo può essere particolarmente dannoso in applicazioni critiche, dove dati sensibili o comandi di controllo potrebbero essere intercettati o alterati da attori malintenzionati.

Eseguire algoritmi pesanti di cifratura su microcontrollori con risorse limitate di calcolo e memoria può rivelarsi impraticabile o sconveniente, essendo dispositivi i dispositivi IoT dotati di poca capacità computazionali e di memoria ristrette, rendendo difficile l'implementazione di tecniche crittografiche avanzate senza compromettere le prestazioni o la funzionalità.

La cifratura, sebbene essenziale per la sicurezza, richiede un equilibrio tra il livello di sicurezza desiderato e le capacità del dispositivo. Pertanto, l'adozione di metodi di cifratura leggeri o l'ottimizzazione delle risorse disponibili diventa cruciale per mantenere sia la sicurezza che l'efficienza operativa dei microcontrollori in contesti a risorse limitate.

Nel caso di mancanza di cifratura dei dati trasmessi attraverso MQTT rende il protocollo vulnerabile agli attacchi di intercettazione. Senza cifratura, i dati inviati tra

i dispositivi e il broker possono essere facilmente letti da chiunque si trovi sulla stessa rete, compromettendo la confidenzialità delle informazioni scambiate. Un altro problema significativo è la gestione inadeguata dei topic (Andy, 2017).

In MQTT, i topic sono utilizzati per filtrare e indirizzare i messaggi ai sottoscrittori appropriati. Tuttavia, senza una gestione appropriata dei permessi, è possibile che utenti non autorizzati sottoscrivano a topic sensibili, ottenendo così accesso a dati che non dovrebbero essere a loro disposizione.

L'uso delle wildcard nei sistemi di messaggistica presenta potenziali rischi per la sicurezza e la gestione delle informazioni, benché le wildcard offrano flessibilità nelle sottoscrizioni ai topic, consentendo ai client di ricevere messaggi da più fonti con una singola sottoscrizione, questa caratteristica può esporre i sistemi a vari pericoli. La sottoscrizione a topic troppo ampi può portare alla ricezione di dati non desiderati o sensibili, aumentando il rischio di sovraccarico del sistema o di violazioni della privacy. Inoltre, l'uso improprio delle wildcard potrebbe facilitare attacchi mirati alla rete, come l'intercettazione o il flooding di messaggi. È fondamentale, quindi, adottare politiche di sicurezza stringenti, limitare l'uso delle wildcard a casi d'uso ben definiti e monitorare attentamente l'accesso ai topic per mitigare questi rischi e garantire la sicurezza dei dati trasferiti. Questo rischio è amplificato dalla tendenza di alcuni sviluppatori a utilizzare schemi di denominazione prevedibili per i topic, rendendo semplice per gli attaccanti indovinare i nomi dei topic a cui sottoscrivere. La resilienza agli attacchi di tipo Denial of Service (DoS) è un altro problema. Dato che MQTT si basa su una connessione persistente tra il dispositivo e il broker, un attaccante potrebbe facilmente sovraccaricare il sistema con richieste malevole, rendendo il servizio indisponibile per gli utenti legittimi. Questo tipo di attacco può avere conseguenze devastanti, soprattutto in sistemi critici dove la disponibilità continua dei dati è essenziale (Andy, 2017).

Da un punto di vista locale, un attaccante può intercettare e alterare i dati scambiati nella rete, compromettendo così la privacy, l'integrità dei dati e i meccanismi di autenticazione del protocollo MQTT. È da notare che l'impiego di porte non convenzionali non incrementa la sicurezza di MQTT. Per mitigare tali vulnerabilità, è fondamentale implementare meccanismi di sicurezza robusti per MQTT. L'uso di TLS rappresenta una soluzione efficace, specialmente per dispositivi IoT non soggetti a restrizioni di risorse. In aggiunta, Singh et al. hanno suggerito un approccio di sicurezza basato sull'algoritmo di crittografia ECC, vantaggioso per la sua minore

richiesta di risorse rispetto a TLS e focalizzato sulla protezione della riservatezza dei dati (Andy, 2017).

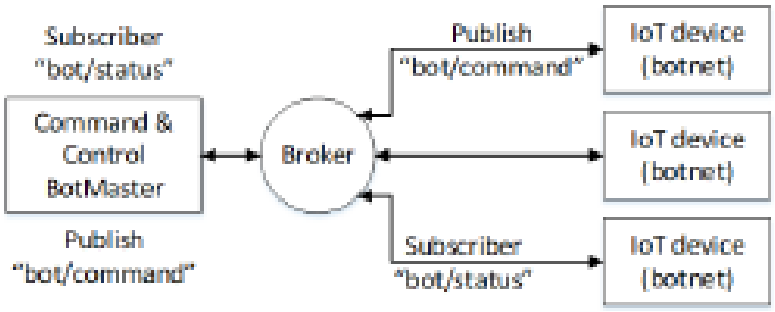

Figura 5. Scenario di comando e controllo di botnet tramite MQTT

Analogamente, Mektoubi et al., (2016) hanno condotto un'analisi comparativa tra RSA ed ECC, evidenziando come quest'ultimo garantisca una migliore protezione dei dati e assicuri affidabilità nella non ripudiabilità. Per dispositivi con capacità limitate, Niruntasukrat et al. hanno sviluppato un meccanismo di sicurezza incentrato sull'autenticazione e autorizzazione dei dispositivi, mentre Katsikeas ha proposto l'uso della crittografia AES, mirata alla salvaguardia della riservatezza e autenticità dei messaggi. Nonostante questi avanzamenti, la sicurezza del protocollo MQTT, in particolare per dispositivi a risorse ridotte, necessita di ulteriori ricerche e sviluppi. Attualmente, ogni soluzione proposta si concentra su aspetti specifici senza offrire un approccio olistico alla sicurezza.

CAPITOLO III

INSTRUMENTAZIONE BINARIA

3.1 Introduzione e definizione dell'instrumentazione binaria

L'instrumentazione binaria rappresenta un concetto fondamentale nel campo dell'ingegneria del software e della sicurezza informatica, che si riferisce alla tecnica di analisi e modifica di programmi eseguibili senza accedere al loro codice sorgente originale. Questa pratica consente agli sviluppatori e agli analisti di inserire codice aggiuntivo o di modificare il comportamento esistente di un'applicazione al fine di monitorare o cambiare la sua esecuzione in tempo reale. L'obiettivo principale dell'instrumentazione binaria è quello di fornire una comprensione profonda del comportamento del software in vari scenari di esecuzione, migliorando la sicurezza, le prestazioni e la qualità del software. L'instrumentazione binaria può essere ottenuta in due modi (Du, 2012):

1. **Instrumentazione binaria dinamica (DBI):** instrumentazione just-in-time del codice in esecuzione. Questo permette di analizzare e modificare il comportamento del programma mentre è in esecuzione.
2. **Instrumentazione binaria statica (SBI):** si inserisce del codice aggiuntivo direttamente nel file eseguibile, permettendo così di monitorare il comportamento del programma.

L'instrumentazione binaria statica consiste nell'analizzare il programma prima del tempo di esecuzione, disassemblando prima il codice e successivamente rilevando i punti nel programma che richiedono l'instrumentazione andando a modificare permanentemente il binario. Questi punti (chiamati punti di instrumentazione) verranno utilizzati come posizioni per l'inserimento del codice aggiuntivo (chiamato codice di instrumentazione) (Figura 6) (Zhang, 2014).

D'altra parte, la instrumentazione binaria dinamica si basa su un monitor esterno che monitora il programma al momento dell'esecuzione. Il monitor controlla un'istruzione alla volta prima di essere eseguita e inserisce al volo il codice di strumentazione. I principali vantaggi del DBI sono:

a) si evita il sovraccarico in termini di costi di memoria, poiché l'eseguibile originale viene mantenuto inalterato;

b) la strumentazione viene eseguita al momento dell'esecuzione, senza la necessità di un costoso processo offline.

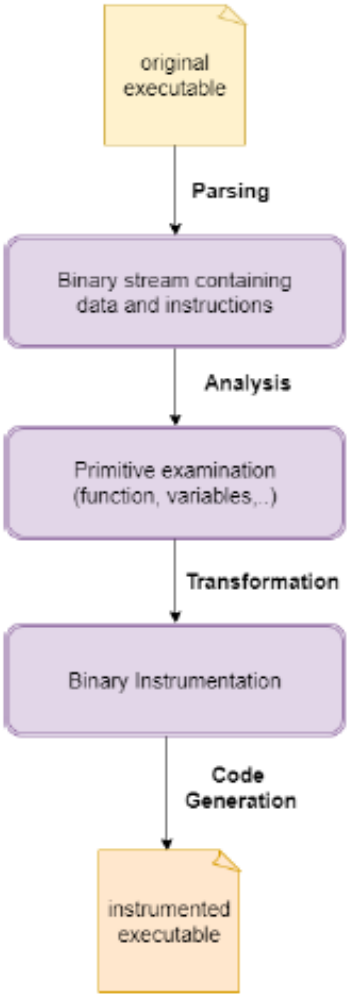

Figura 6. Processo di instrumentazione binaria statica

Inoltre, l'approccio in tempo reale adottato da DBI non deve preoccuparsi della rilocazione del codice, che deve essere affrontata da SBI durante la fase di trasformazione. DBI ha invece i suoi svantaggi nell'overhead introdotto in termini di tempo di esecuzione, che può incidere significativamente sulle prestazioni. Questa è una caratteristica particolarmente critica nelle applicazioni in tempo reale in sistemi embedded.

SBI introduce un sovraccarico in termini di prestazioni, ma è dato solo dalle istruzioni di instrumentazione qualora vengano inserite, cioè minimo rispetto a quanto accade con DBI, dove avviene un'esecuzione per simboli. Con entrambe le strategie, il processo di instrumentazione segue le seguenti fasi (Wenzl, 2019):

1. Parsing: lo scopo di questa fase è ottenere il flusso di istruzioni grezze dall'eseguibile e quindi inviarlo come input a un disassemblatore. Inoltre, il contenuto delle variabili globali e della sezione dati viene raccolto per ulteriori analisi;
2. Analisi: in questa fase viene recuperata la struttura del programma. Il flusso di istruzioni è disassemblato e le istruzioni disassemblate sono raggruppate in funzioni. Inoltre, vengono rilevate le istruzioni condizionali e viene prodotto un grafico del flusso di controllo (CFG);
3. Trasformazione: una volta disponibile il CFG, i punti di instrumentazione nel codice vengono raccolti e integrati con il codice di strumentazione;
4. Generazione del codice: è l'ultima fase del processo, in cui le modifiche vengono integrate nel programma in modo da produrre un eseguibile finale.

3.2 Applicazioni nell'ambito del monitoraggio

Una delle applicazioni primarie dell'instrumentazione binaria si trova nel monitoraggio delle prestazioni del software. Attraverso questa tecnica, gli sviluppatori possono inserire contatori di prestazioni o timer in punti specifici di un programma per raccogliere informazioni dettagliate sull'esecuzione, come il tempo impiegato da particolari funzioni o il numero di volte che un certo blocco di codice viene eseguito. Questi dati possono essere poi analizzati per identificare colli di bottiglia nelle prestazioni e guidare l'ottimizzazione del codice. Nell'ambito della sicurezza informatica, l'instrumentazione binaria è utile agli sviluppatori perché consente di iniettare nel codice dei checkpoint o dei trigger in punti strategici, che possono rivelare o bloccare comportamenti sospetti o non autorizzati. Attraverso l'instrumentazione binaria, è possibile monitorare il flusso di esecuzione del programma e intercettare tentativi di exploit, accessi non autorizzati o altre anomalie, contribuendo significativamente al rafforzamento delle misure di protezione del software. Inoltre, l'instrumentazione binaria può essere utilizzata per implementare controlli di sicurezza aggiuntivi che verificano l'integrità dei dati in esecuzione o per identificare tentativi di exploit derivanti da overflow di buffer, iniezioni di codice o altri vettori di attacco. Integrando questi controlli direttamente nel codice binario, gli sviluppatori hanno la capacità di proteggere le applicazioni anche in assenza del codice sorgente originale o in contesti in cui il sorgente non è modificabile. Questo approccio rende l'instrumentazione binaria uno strumento estremamente potente per migliorare non solo le prestazioni ma anche la sicurezza dei software. Le soluzioni di integrità del

flusso di controllo basate su software si basano su monitor del flusso di controllo implementati esclusivamente utilizzando tecniche software. In altre parole, i controlli vengono eseguiti tramite codice aggiuntivo, eseguito in parallelo attraverso un altro processo o all'interno del programma stesso, tramite strumentazione binaria (Zhang, 2014).

Nell'articolo originale di Abadi (2005) che descrive CFI, gli autori propongono una instrumentazione basata su etichette per fare in modo che il software controlli da solo la validità del flusso di controllo. L'approccio si basa sulla memorizzazione di identificatori univoci (etichette) all'inizio di ciascun blocco di base, dove un blocco di base in un CFG corrisponde a un insieme di istruzioni senza istruzioni di salto in mezzo. Il codice di instrumentazione viene inserito prima del ramo indiretto e controlla se l'ID del blocco base di destinazione corrisponde a quello atteso recuperato dall'analisi CFG. Nel caso di tentativi malintenzionati di controllo del flusso di esecuzione, il tentativo viene rilevato e l'integrità del flusso di esecuzione è garantita. Nella Figura 7 è mostrato un esempio di come funziona il meccanismo (Abadi, 2005).

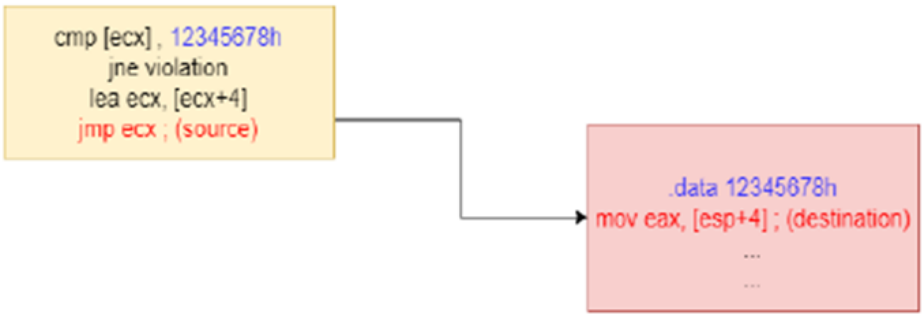

Figura 7. Codice della strumentazione basato su etichetta

In questo caso, la destinazione del salto è stata etichettata con l'ID 12345678. Prima di eseguire JMP ECX, l'etichetta di destinazione viene controllata per rilevare tentativi di manomissione del flusso di controllo. Nell'esempio se il controllo fallisce, l'esecuzione viene reindirizzata alla funzione penetration(), che fermerà l'esecuzione; in caso contrario, ECX viene incrementato e l'esecuzione viene reindirizzata con successo alla destinazione corretta (Abadi, 2005).

Un'altra applicazione significativa dell'instrumentazione binaria è nel campo del reverse engineering e dell'analisi malware. Il reverse engineering del malware esegue l'analisi dinamica del codice per ispezionare un programma durante l'esecuzione. Ciò in genere comporta l'utilizzo di un debugger per monitorare un processo sospetto. Un

approccio complementare consiste nell'interrogare un processo in esecuzione utilizzando i framework DBI. Mentre un debugger consente di collegarsi a un processo, le tecniche DBI consentono di inserire ed eseguire codice all'interno di un processo per esaminarne gli aspetti interni. I framework DBI più noti includono Pin, DynamoRIO e Frida di Intel. Questi framework vengono spesso utilizzati per valutare programmi proprietari e valutare le prestazioni dei programmi, ma possono anche essere applicati per accelerare l'analisi del malware. Consentono agli analisti di agganciare funzioni per osservare le chiamate API, valutare i loro input e output e persino modificare istruzioni e dati durante l'esecuzione. I framework DBI sono destinati sia ai sistemi operativi desktop che mobili (ad esempio Windows, macOS, GNU/Linux, iOS, Android™ e QNX) e forniscono API ben documentate per facilitare lo sviluppo degli strumenti. Gli analisti di sicurezza utilizzano spesso questa tecnica per inserire punti di traccia o di interruzione nel codice di programmi sospetti senza alterarne il comportamento osservabile. Ciò permette di monitorare l'esecuzione del malware in ambienti controllati, analizzando le sue interazioni con il sistema operativo e con la rete, per comprendere meglio le sue funzionalità e sviluppare contromisure efficaci (Blogs, 2021).

Inoltre, l'instrumentazione binaria trova impiego nel testing e nella verifica del software, consentendo agli sviluppatori di inserire automaticamente asserzioni e controlli di validità nel codice binario per rilevare errori di programmazione, di gestione della memoria, o altri tipi di bug durante l'esecuzione del programma. Questo approccio può significativamente aumentare la copertura dei test, fornendo una verifica più estesa rispetto ai metodi tradizionali basati sul solo codice sorgente. Complessivamente, l'instrumentazione binaria offre una gamma estremamente versatile di applicazioni che spaziano dal miglioramento delle prestazioni e della sicurezza dei software fino alla facilitazione della ricerca e dell'analisi di errori. La sua capacità di modificare e monitorare i programmi a livello binario apre nuove possibilità nel campo dell'ingegneria del software, rendendola una tecnica preziosa per sviluppatori, analisti di sicurezza, e ricercatori (Brignon, 2019).

3.3 Strumenti e tecnologie per l'instrumentazione binaria

In letteratura sono stati presentati diversi strumenti di strumentazione binaria, basati sui vantaggi e sugli svantaggi della tecnica di strumentazione. Nella presente sezione verranno brevemente presentati alcuni degli strumenti più comuni che seguono SBI o DBI.

- PEBIL (PMaC’s Efficient Binary Instrumentation Toolkit for Linux) è un toolkit di strumentazione binaria che consente la creazione di strumenti di strumentazione producendo eseguibili strumentati efficienti, sfruttando l’approccio statico. PEBIL strumenta l'eseguibile staticamente inserendo un'istruzione di salto nel punto di strumentazione che reindirizzerà l'esecuzione al codice di strumentazione. Il codice di strumentazione salva lo stato del programma ed eseguirà le azioni richieste dalla strumentazione. Per supportare gli ISA con istruzioni di lunghezza variabile, PEBIL si occupa di riposizionare e trasformare il codice per ciascuna funzione al fine di rendere il codice di strumentazione raggiungibile dai punti di strumentazione. PEBIL supporta solo l'architettura x86 (Laurenzano, 2010).
- PIN: come framework di strumentazione binaria dinamica sviluppato da Intel. Pin mira a fornire una piattaforma per la creazione di strumenti di analisi dei programmi, chiamati pintools. Un pintool è costituito da tre tipi di routine:
    - routine di strumentazione: routine che controllano le istruzioni dell'applicazione e inseriscono chiamate alle routine di analisi;
    - routine di analisi: invocate quando il programma esegue un'istruzione strumentata;
    - routine di richiamata: richiamate quando si verifica un evento;

  Le API C/C++ fornite da PIN possono essere sfruttate per creare pintools. PIN utilizzerà un compilatore Just-In-Time (JIT) per applicare la strumentazione in fase di esecuzione all'applicazione. Inoltre, il PIN supporta l'opzione della modalità sonda, che fornisce prestazioni più elevate ma ha un set limitato di callback disponibili che limitano le capacità del pintool. PIN è fornito per piattaforme Windows e Linux, ma funziona solo per programmi compilati per l'architettura Intel, rendendolo non adatto a sistemi embedded basati su architetture ARM (Bach, 2010).
- Dynist è un framework di strumentazione e analisi binaria che sfrutta sia l'approccio statico che quello dinamico per la strumentazione del codice. Dynist fornisce una rappresentazione del programma in termini di strutture di livello più elevato, come i blocchi base della funzione, e l'utente sfrutta questa rappresentazione per strumentare il codice in qualsiasi punto del binario. Dynist enfatizza la strumentazione in qualsiasi momento fornendo la possibilità di strumentare staticamente il codice (binario modificato in modo

permanente) o di strumentarlo al momento dell'esecuzione (strumentazione dinamica). L'utente può modificare o rimuovere la strumentazione in qualsiasi momento. Inoltre, le tecniche di analisi eseguite consentono di inserire nuove istruzioni senza influenzare il codice non strumentato, che può funzionare a velocità nativa, in modo da minimizzare l'overhead nelle prestazioni (Bernat, 2011).

3.4 Falco: Panoramica e applicazioni nella Sicurezza IoT

In un'era digitale caratterizzata da una rapida evoluzione delle minacce informatiche, emerge l'importanza di strumenti sofisticati di sicurezza, tra cui Falco, un'iniziativa open source sviluppata originariamente da Sysdig, e ora facente parte del prestigioso Cloud Native Computing Foundation (CNCF). La sua essenza, profondamente radicata nella protezione runtime cloud-native, si manifesta attraverso la sua abilità unica di scrutare, con occhio vigile, comportamenti fuori dall'ordinario e potenziali insidie che minacciano l'integrità dei sistemi, con un focus particolare sugli ambienti Linux (Falco, 2024).

Falco funziona intercettando e analizzando le chiamate di sistema attraverso una tecnica nota come "system call tracing", ed eseguendo una serie di regole che permetto la rilevazione di anomalie (figura 8). Falco semplifica il monitoraggio delle chiamate di sistema del kernel Linux e arricchisce tali eventi con informazioni provenienti da Kubernetes e dal resto dello stack nativo del cloud. Falco dispone di un ricco set di regole di sicurezza pronte all'uso create appositamente per Kubernetes, Linux e il cloud. Dalla sua nascita, Falco è stato scaricato più di 100 milioni di volte, con una crescita di oltre il 480% negli ultimi due anni.

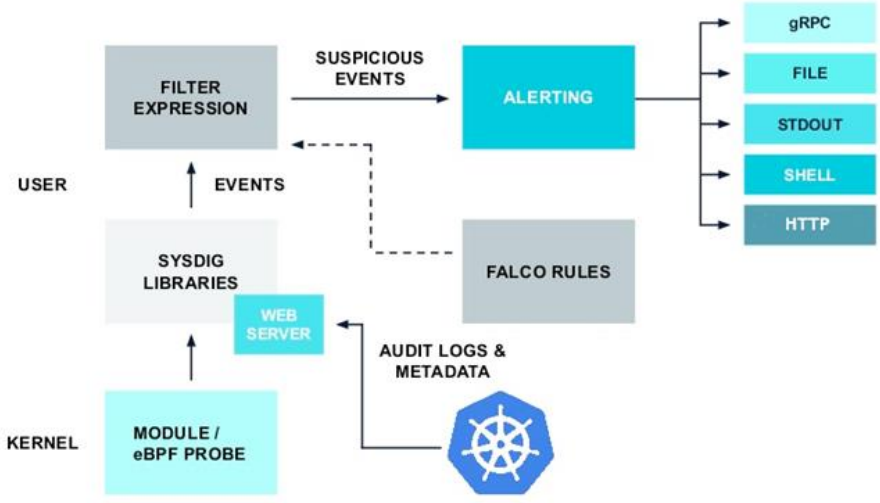

Figura 8. Architettura di Falco

3.4.1 Descrizione di Falco

Falco è uno strumento di sicurezza runtime progettato per monitorare e proteggere i sistemi operativi Linux da comportamenti anomali e potenziali minacce alla sicurezza in tempo reale, questo lo rende a tutti gli effetti uno strumento di instrumentazione binaria dinamica (BDI). Falco è ampiamente utilizzato in ambienti di produzione da varie organizzazioni. Ad esempio, GitLab, una nota piattaforma per la gestione del ciclo di vita del software, utilizza Falco per rilevare attività sospette e garantire la sicurezza delle loro infrastrutture e dei loro servizi, allo stesso modo, Shopify, una delle principali piattaforme di e-commerce, si affida a Falco per proteggere i dati sensibili dei loro clienti e prevenire possibili violazioni della sicurezza. La sua funzionalità principale si basa sull'osservazione degli eventi del sistema, come le chiamate di sistema (syscalls), e sull'integrazione di metadati dal runtime dei container e da Kubernetes (Falco, 2024).

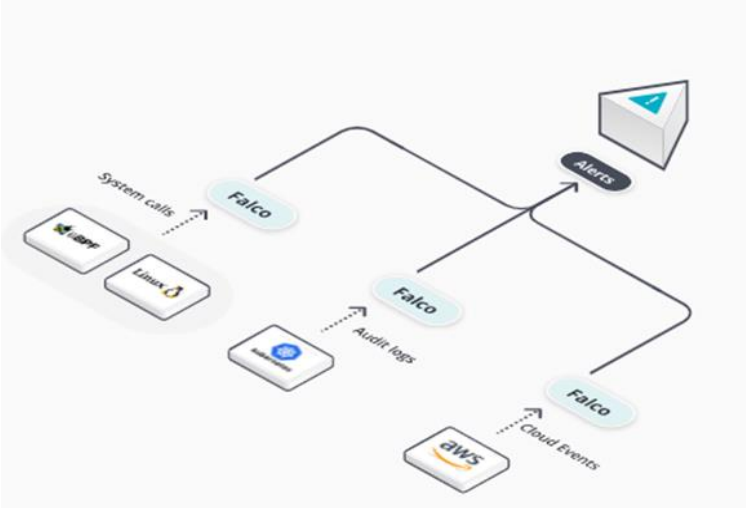

Figura 9. Ambito di operazione di Falco

Falco opera sia nel kernel che nello spazio utente. Nello spazio del kernel, le chiamate di sistema Linux (syscalls) vengono raccolte da un driver, ad esempio il modulo del kernel Falco o la sonda eBPF Falco. Successivamente, le chiamate di sistema vengono posizionate in un ring buffer da cui vengono spostate nello spazio utente per l'elaborazione. Gli eventi vengono filtrati utilizzando un motore di regole con un set di regole Falco. Falco viene fornito con un set di regole predefinito, ma gli operatori possono modificare o disattivare tali regole e aggiungerne di proprie. Se Falco rileva eventi sospetti, questi vengono inoltrati a vari endpoint.

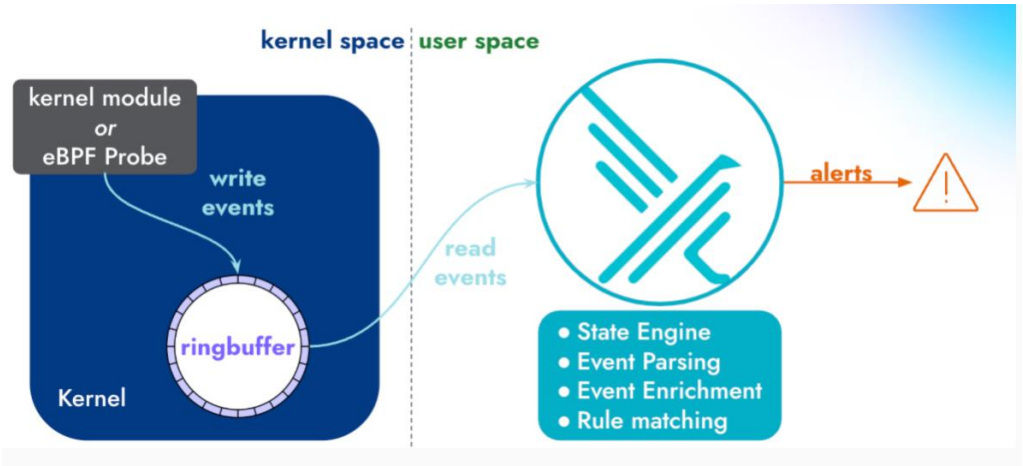

Figura 10. Metodo di gestione delle chiamate di Falco

3.4.2 Funzionalità chiave e casi d'uso di Falco

Le principali funzionalità di Falco possono essere così riassunte (Falco, 2024):

- Monitoraggio in tempo reale: Falco monitora le chiamate di sistema nel kernel Linux in tempo reale, offrendo una visione approfondita dell'attività del sistema. Nel contesto della sicurezza informatica e del monitoraggio dei sistemi, gli strumenti come Falco, sviluppato dalla Sysdig, sono progettati per rilevare comportamenti anomali o potenzialmente dannosi all'interno dei sistemi operativi, principalmente attraverso l'osservazione delle chiamate di sistema (syscalls). Le chiamate di sistema sono fondamentali per l'interazione tra il software in esecuzione e il nucleo del sistema operativo, permettendo di eseguire operazioni a basso livello come la lettura o scrittura di file, la gestione della memoria e la comunicazione di rete.
- Regole personalizzabili: Gli utenti possono definire regole specifiche che descrivono comportamenti indesiderati o potenzialmente pericolosi, permettendo a Falco di generare allarmi quando queste condizioni vengono soddisfatte.
- Estensibilità tramite plugin: L'architettura di Falco supporta plugin che estendono le sue capacità di monitoraggio oltre le syscalls, permettendo di includere fonti di eventi aggiuntive come log di audit di Kubernetes, eventi CloudTrail di AWS, e molto altro. Falco offre un'architettura estensibile che permette di monitorare una vasta gamma di eventi, non limitandosi esclusivamente alle chiamate di sistema (syscalls). Questa estensibilità è resa

possibile grazie all'uso di plugin e alla possibilità di integrare specifici "probe" nel codice che si desidera monitorare.

- Integrazione con l'ecosistema Cloud-Native: Falco si integra strettamente con tecnologie cloud-native come Kubernetes, offrendo capacità di monitoraggio e allerta specifiche per queste piattaforme.
- Output e allerte configurabili: Gli allarmi generati da Falco possono essere inviati a vari destinatari e formati, inclusi log standard, syslog, endpoint HTTP[S], e API gRPC, facilitando l'integrazione con sistemi di gestione degli eventi e di risposta agli incidenti.

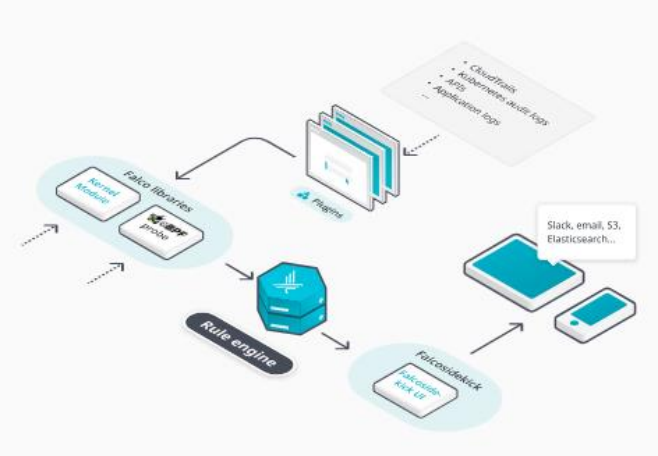

Figura 11. Come lavora Falco

L'osservazione delle chiamate di sistema è fondamentale per le prestazioni e ci sono due modi in cui Falco raggiunge questo obiettivo: una sonda eBPF o un modulo del kernel. eBPF è una tecnologia rivoluzionaria che ci consente di eseguire programmi sandbox all'interno di un sistema operativo. Gli script eBPF sono flessibili, sicuri e funzionano estremamente velocemente, rendendoli perfetti per acquisire la sicurezza del runtime. Ciò lo rende ideale per le chiamate di sistema di strumentazione per Falco. Prima che emergesse eBPF, i moduli del kernel erano la norma per estendere le funzionalità nel kernel Linux. Funzionano in modalità privilegiata e sono scritti in C, il che li rende efficienti e un'opzione eccellente per lavori critici in termini di prestazioni. Falco offre un modulo kernel per le situazioni in cui eBPF non è la soluzione migliore (Falco, 2024).

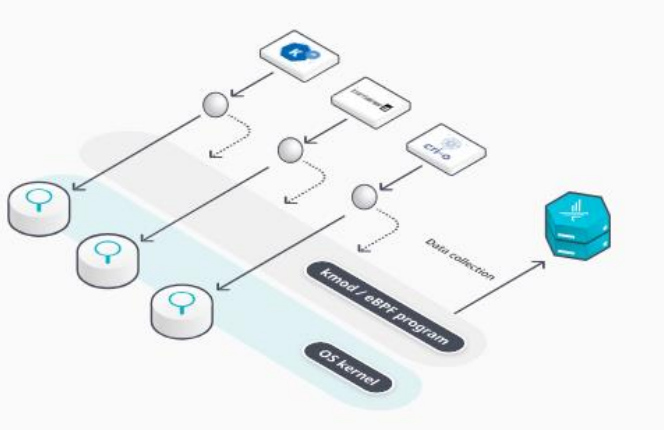

Figura 12. Sistema dell'instrumentazione delle chiamate

A differenza dei moduli del kernel, che richiedono codice a livello di kernel per essere compilato e caricato direttamente nel kernel, eBPF funziona in uno spazio isolato all'interno del kernel, offrendo un ambiente di esecuzione sicuro e controllato per il codice di monitoraggio e di manipolazione dei dati. Uno dei principali vantaggi di eBPF è la sicurezza. Poiché il codice eBPF viene eseguito in un ambiente virtuale controllato, con accesso limitato alle risorse del kernel, il rischio di crash del sistema o di comportamenti indesiderati è significativamente ridotto rispetto ai moduli del kernel. Inoltre, eBPF impone un controllo rigoroso del codice prima dell'esecuzione, assicurando che solo codice verificato e sicuro possa essere eseguito. Dal punto di vista dell'efficienza, eBPF è progettato per essere estremamente performante. Il codice eBPF viene compilato in bytecode e poi ottimizzato dinamicamente dal Just-In-Time compiler del kernel, risultando in esecuzioni molto veloci e con un basso overhead. Questo lo rende ideale per il monitoraggio in tempo reale e l'analisi delle prestazioni senza impattare significativamente le prestazioni del sistema. La flessibilità è un altro vantaggio chiave di eBPF. Consente agli sviluppatori di scrivere programmi che possono essere eseguiti in vari punti di aggancio all'interno del kernel, fornendo la capacità di monitorare e influenzare praticamente qualsiasi aspetto del comportamento del sistema (Scholz, 2018).

Con Falco e Falcosidekick è anche possibile inoltrare eventi sospetti a sistemi serverless per attivare azioni e porre rimedio alle minacce. Falcosidekick è un'applicazione complementare a Falco che inoltra gli eventi Falco. Consente di distribuire eventi a più di 50 sistemi, come e-mail, chat, code di messaggi, funzioni serverless, database e altro ancora. È facile da configurare e utilizzare sia localmente che all'interno di Kubernetes (Falco, 2024).

Figura 13. Metodo di reazione alle minacce

Falco, essendo uno strumento versatile e potente per il monitoraggio della sicurezza in tempo reale, trova applicazione in una vasta gamma di scenari nel mondo reale. Questi casi d'uso dimostrano come Falco possa essere impiegato per rafforzare la sicurezza in diversi ambienti e contesti. Ecco alcuni esempi concreti di come Falco viene utilizzato per (Falco, 2024):

- Monitoraggio di ambienti ubernetes: In un'organizzazione che utilizza Kubernetes per gestire le sue applicazioni containerizzate, Falco può essere configurato per monitorare e allertare su eventi sospetti o non conformi alle politiche di sicurezza aziendali. Ad esempio, può rilevare tentativi di accesso non autorizzati a pod Kubernetes sensibili, modifiche inaspettate alle configurazioni dei container o tentativi di escalation di privilegi all'interno del cluster. Questo aiuta i team di sicurezza a intervenire rapidamente per mitigare potenziali minacce.
- Protezione delle infrastrutture cloud: Un'azienda che opera nel cloud può utilizzare Falco per monitorare le attività sospette nelle sue istanze di calcolo, come accessi anomali o modifiche non autorizzate ai file di configurazione critici. Falco può essere configurato per generare allarmi in tempo reale quando vengono rilevate attività che potrebbero indicare una compromissione, consentendo ai team di sicurezza di indagare e rispondere prontamente.
- Conformità e auditing in ambienti regolamentati: In settori fortemente regolamentati, come il finanziario, la sanità o il pubblico, Falco può essere impiegato per garantire che le attività di sistema rispettino le normative sulla

privacy e sulla protezione dei dati. Tracciando l'accesso ai dati sensibili e registrando le attività di sistema, Falco fornisce un meccanismo efficace per l'auditing e la dimostrazione della conformità alle normative, come GDPR, HIPAA o PCI-DSS.

- Rilevamento di anomalie in tempo reale: Un provider di servizi Internet può utilizzare Falco per monitorare i suoi server per rilevare comportamenti anomali che potrebbero indicare un attacco DDoS, tentativi di intrusione o malware. Configurando regole specifiche, Falco può identificare pattern di traffico insoliti o attività di sistema sospette, permettendo al team di sicurezza di agire rapidamente per mitigare l'attacco e mantenere la continuità del servizio.
- Integrazione con sistemi di risposta agli incidenti: Falco può essere integrato con piattaforme di orchestrazione della risposta agli incidenti (SOAR) e sistemi di gestione degli eventi di sicurezza e informazioni (SIEM) per automatizzare la risposta agli allarmi di sicurezza. In un'organizzazione che dispone di un centro operativo di sicurezza (SOC), gli allarmi di Falco possono essere automaticamente inviati al SIEM per l'analisi e, se necessario, scatenare workflow automatizzati nel SOAR per isolare dispositivi compromessi, bloccare indirizzi IP sospetti o eseguire altre azioni di mitigazione.
- Sicurezza IoT e monitoraggio dei dispositivi: In un contesto IoT, dove i dispositivi connessi sono spesso distribuiti su larga scala e in ambienti critici, Falco può essere utilizzato per monitorare il comportamento di questi dispositivi. Rilevando anomalie come tentativi di accesso non autorizzati, modifiche inaspettate alla configurazione o comunicazioni di rete sospette, Falco aiuta a proteggere l'ecosistema IoT da potenziali vulnerabilità e attacchi.

3.4.3 Falco per il rafforzamento della sicurezza dei sistemi IoT

Nel contesto della sicurezza IoT, l'importanza di strumenti come Falco diventa sempre più evidente man mano che il numero e la complessità dei dispositivi connessi continua a crescere. Questi dispositivi, spesso distribuiti su vasta scala e in ambienti critici, possono diventare bersagli critici per gli attacchi informatici. In questo scenario, Falco emerge come uno strumento utile per rafforzare la sicurezza dei dispositivi e delle infrastrutture IoT, grazie alla sua capacità di monitoraggio in tempo reale. Falco offre agli amministratori di sistema la possibilità di identificare in tempo reale attività sospette o anomalie che potrebbero indicare una compromissione della sicurezza.

Questo include il rilevamento di tentativi di escalation di privilegi, che potrebbero indicare un attacco in corso per ottenere un controllo più profondo sul sistema, o accessi non autorizzati a file e directory critici, che potrebbero suggerire tentativi di furto di dati o di manipolazione del sistema. Implementando regole personalizzate in Falco, gli operatori possono definire quali comportamenti considerare pericolosi o non autorizzati, consentendo un monitoraggio mirato e altamente efficace (Falco, 2024).

Un elemento chiave di Falco è la sua capacità di utilizzare regole personalizzate, che consentono agli utenti di definire comportamenti specifici da monitorare. Questa flessibilità nelle regole di configurazione aumenta significativamente la granularità del monitoraggio, permettendo una sorveglianza dettagliata e mirata che è cruciale in ambienti IoT, dove ogni dispositivo può avere esigenze di sicurezza diverse. Falco è quindi utile per identificare configurazioni errate, accessi non autorizzati o esecuzione di comandi insoliti, contribuendo a proteggere le infrastrutture IoT da potenziali minacce. Questa capacità di introspezione assicura la sicurezza e l'integrità delle infrastrutture.

La capacità di Falco di generare allarmi dettagliati e configurabili lo rende un candidato ideale per l'integrazione con sistemi di risposta automatica agli incidenti. Questi allarmi possono innescare azioni correttive immediate, come l'isolamento di dispositivi compromessi per prevenire ulteriori danni. Questo tipo di integrazione consente una risposta rapida agli incidenti, riducendo il tempo di esposizione a potenziali minacce e minimizzando l'impatto degli attacchi. Le normative in materia di sicurezza e privacy dei dati stanno diventando sempre più stringenti, soprattutto in settori critici come quello sanitario, finanziario e dei servizi pubblici, dove i dispositivi IoT trovano ampio impiego. Le capacità di monitoraggio e registrazione di Falco aiutano a soddisfare questi requisiti di conformità e auditing dei sistemi IoT, fornendo registrazioni dettagliate delle attività di sistema e degli eventi di sicurezza. Questo non solo permette di dimostrare la conformità con le normative vigenti ma offre anche preziose informazioni forensi in caso di incidenti di sicurezza (Falco, 2024).

3.5 RGB565

Il formato RGB565 rappresenta un metodo efficiente e diffuso per codificare i colori in dispositivi digitali, specialmente in quelli con limitazioni di memoria o di larghezza di banda. Questo formato utilizza 16 bit per rappresentare un singolo colore, distribuiti tra i canali red, green e blue, con una distribuzione di 5 bit per il red, 6 bit per il green e 5 bit per il blue. La scelta di questa distribuzione di bit prende in considerazione la

maggiore sensibilità dell'occhio umano alle variazioni di verde, permettendo così una rappresentazione più accurata dei colori pur mantenendo un formato compatto (RGB565 Color Picker, 2024).

Tra i vantaggi possiamo menzionare (RGB565 Color Picker, 2024):

- Efficienza di memoria: Utilizzando solo 16 bit per colore, il formato RGB565 consente di ridurre significativamente l'uso della memoria, rispetto ai 24 bit richiesti dal formato RGB888 più diffuso. Questo si traduce in un risparmio del 33% di spazio, fondamentale in dispositivi con risorse limitate.
- Prestazioni migliorate: La riduzione della quantità di dati da elaborare può portare a un miglioramento delle prestazioni in applicazioni in tempo reale, come i videogiochi o il rendering di grafica animata, dove la velocità di elaborazione è essenziale.
- Consumo energetico ridotto: In dispositivi portatili, dove il risparmio energetico è fondamentale per prolungare la durata della batteria, l'uso di un formato di colore più efficiente può contribuire a ridurre il consumo energetico complessivo.

Per gli svantaggi, invece (RGB565 Color Picker, 2024):

- Profondità di colore limitata: Con solo 65.536 colori disponibili, il formato RGB565 può risultare limitato per applicazioni che richiedono una vasta gamma cromatica o che devono gestire sfumature delicate. Questo può portare a effetti indesiderati come il banding, dove le transizioni di colore non sono più fluide.
- Assenza di trasparenza: A differenza di altri formati che includono un canale alfa per gestire la trasparenza, il formato RGB565 non supporta nativamente questa caratteristica, limitando l'uso in contesti dove la trasparenza non è necessaria.

La conversione da RGB888 a RGB565 e viceversa è un'operazione comune in molte applicazioni grafiche. Questo processo implica una riduzione o un'estensione dei valori di colore per adattarli al nuovo formato (RGB565 Color Picker, 2024).

CAPITOLO IV

CONTRIBUTO DELLA RICERCA

Il contributo di questa ricerca si focalizza sull'utilizzo di tecniche, tra cui l'instrumentazione binaria, per il monitoraggio a runtime di dispositivi IoT e sull'indagine del corrispondente overhead introdotto.

Attraverso una valutazione quantitativa, vengono esaminati l'impatto e le metriche specifiche dell'overhead su prestazioni e sicurezza. La ricerca include test e benchmark per quantificare l'overhead in vari scenari.

Viene inoltre offerta una guida pratica per l'integrazione di sistemi di monitoraggio, Il lavoro contribuisce alla comprensione dell'interazione tra sicurezza e prestazioni nei sistemi IoT.

4.1 Analisi degli attori e architettura del sistema

La struttura del sistema in esame è descritta con un dettagliato Deployment diagram UML (figura 14) che illustra le varie componenti e le loro interazioni all'interno della rete.

Specificatamente, il sistema è composto da cinque principali nodi hardware:

- **Raspberry Pi 4 (RPI4):** Questo nodo funge da fulcro del sistema, ospitando una serie di funzionalità essenziali:
    - L'implementazione di FALCO, approfondita nel capitolo precedente.
      Il suo output viene successivamente instradato verso l'Adapter attraverso un meccanismo di pipe.
    - L'Adapter permette di interfacciare i dati di output di FALCO con il componente MONITOR, convertendoli in una forma più ad alto livello e facilmente utilizzabile per il monitoraggio.
    - Il subscriber 'clientSubSenseHat.js' ha il compito di illuminare i led del Sense-Hat con il colore che riceve attraverso il protocollo MQTT dal client 'clientPub.js'.
- Il **Sense-Hat** è una scheda di espansione progettata per interagire direttamente con il Raspberry Pi mediante l'utilizzo dei pin GPIO e il protocollo I2C.
  Pur essendo dotato di una varietà di sensori, nel nostro esperimento il focus è posto unicamente sulla matrice LED 8x8 del Sense-Hat.

In Linux, questa matrice è trattata come un oggetto di sistema, il che permette al sistema operativo di interagire con essa attraverso un'interfaccia basata su file.

L'uso specifico della syscall ‘pwrite’ consente di scrivere direttamente nel file associato al Sense-Hat. Questo metodo facilita il monitoraggio dei comandi inviati alla matrice tramite scritture.

- **Server HiveMQ**, tale entità opera come broker MQTT.
- **Device 1** il quale ospita:
    - Il client ‘clientPub.js’, ossia un publisher MQTT incaricato di creare e inviare messaggi MQTT che rappresentano i comandi da eseguire sulla matrice LED del Sense-Hat.
- **Device 2,** il quale ospita:
    - Artefatto monitor, sviluppato in Prolog, questo componente riceve i messaggi dall'entità "adapterLight" del Raspberry mediante il protocollo WebSocket e ne verifica la correttezza.

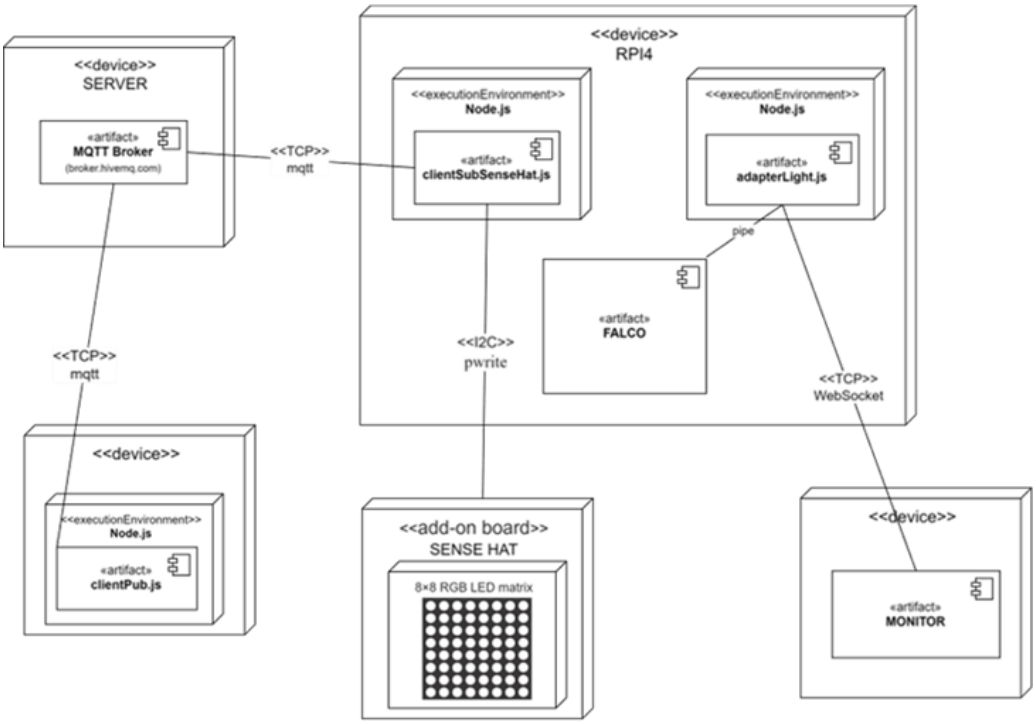

Figura 14. Deployment diagram del sistema in studio

Nel nostro scenario di test, il client ‘clientPub.js’ svolge un ruolo cruciale inviando comandi (semplici messaggi MQTT) dove il topic corrisponde all’ordine da eseguire mentre il payload contiene il colore in formato RGB888. Questi comandi sono destinati al ‘clientSubSenseHat.js’, che ha il compito di interpretarli per controllare la

matrice LED del Sense-Hat. Tuttavia, la scrittura dei valori sul Sense-Hat si realizza in formato esadecimale e quindi in RGB565, introducendo così una complessità aggiuntiva nel processo.

L'obiettivo primario di questo sistema è garantire che le modifiche alla matrice LED del Sense-Hat corrispondano esattamente ai comandi ricevuti. Tale verifica, realizzata dal MONITOR mediante un'analisi incrociata, è cruciale per confermare il corretto funzionamento del sistema, assicurando che ogni istruzione inviata sia interpretata e visualizzata fedelmente sul dispositivo.

4.2 Guida all'uso di Falco

4.2.1 Installazione

Nel caso specifico del nostro studio, l'attenzione è rivolta all'utilizzo di Falco su un nodo Raspberry Pi 4 operante con Ubuntu 20.04, che rappresenta una possibile implementazione in contesti IoT. Come abbiamo già spiegato nel capitolo precedente l'importanza dell'instrumentazione dinamica risiede nella sua capacità di fornire una visuale in tempo reale delle attività a livello di sistema, consentendo la rilevazione e la risposta a potenziali minacce alla sicurezza.

Falco, essendo uno strumento di sicurezza runtime, permette di monitorare e registrare le chiamate di sistema in tempo reale, in questo caso specifico il suo ruolo principale sarà quello di filtrare e registrare solo le TCP syscall provenienti dal broker MQTT e le scritture eseguite sul Sense-Hat.

L'installazione di Falco su distribuzioni Debian-like, come Ubuntu, richiede una serie di passaggi chiave per assicurare un'integrazione corretta e sicura nel sistema. Inizialmente, è cruciale affidare la chiave GPG di falcosecurity, un passo fondamentale per garantire l'autenticità del software installato. Questo si ottiene attraverso il download e l'installazione della chiave GPG, che certifica che i pacchetti ricevuti provengono da una fonte fidata.

Per il nostro caso di studio, abbiamo scelto la modalità kmod per il suo elevato livello di efficienza e precisione nel monitoraggio delle attività a livello di kernel, nonché per la sua capacità di fornire informazioni dettagliate in tempo reale.

Tuttavia, è fondamentale considerare i potenziali svantaggi associati a questo approccio. La stretta associazione di kmod con il kernel host significa che qualsiasi variazione nelle versioni del kernel, nell'architettura o nei sistemi operativi può introdurre complessità significative. Questa interdipendenza richiede che, con ogni

aggiornamento del kernel, il modulo kmod di Falco possa necessitare di una ricompilazione o di adeguamenti per mantenere la compatibilità e garantire prestazioni ottimali. Inoltre, dato che kmod opera a un livello così profondo all'interno del sistema, eventuali malfunzionamenti o bug nel modulo possono influenzare direttamente la stabilità e la sicurezza del sistema host. Pertanto, mentre kmod offre vantaggi incontestabili in termini di capacità di monitoraggio, richiede anche una gestione attenta e proattiva per assicurare che le evoluzioni del sistema non compromettano le funzionalità di Falco (eBPF vs. KMod Performance, 2024).

Un'installazione tipica su Ubuntu richiede l'installazione di dipendenze necessarie per costruire il modulo del kernel, un aspetto fondamentale per utilizzare la modalità kmod di Falco.

4.2.2 Avvio utilizzando regole personalizzate

La configurazione di Falco rappresenta un'operazione critica che incide significativamente sul comportamento del software e sulla sua interazione con il sistema ospitante. Dopo l'installazione, è fondamentale personalizzare il file 'falco.yaml', poiché permette agli amministratori di sistemi di calibrare il monitoraggio in funzione delle caratteristiche dell'ambiente in cui opera il software.

Durante l'inizializzazione, è possibile specificare il file di configurazione da utilizzare, le regole da applicare e la modalità di esecuzione di Falco attraverso l'uso di parametri definiti. Per esempio, il comando da noi utilizzato 'sudo falco -c falco/falco.yaml -r falco/log_server.yaml -r falco/sense-hat.yaml -b -S 400' è l'esempio di un avvio personalizzato, dove si delineano i file di configurazione e le regole specifiche.

Nello specifico, la regola contenuta in 'sense-hat.yaml' si focalizza sul monitoraggio delle operazioni di scrittura 'pwrite' sul file '/dev/fb0', direttamente collegato al display di Sense-Hat, svolgendo un ruolo essenziale nell'identificazione delle interazioni con il dispositivo. Parallelamente, la regola definita in 'log_server.yaml' monitora le syscall TCP associate allo script clientSubSenseHat.js, l'argomento specificato '-S 400' assume una grande importanza perché '-S' ci permette di aumentare le dimensioni massime (in byte) del campo 'env.buffer' di FALCO, nel nostro caso dove in una sola TCP syscall possono trovarsi molteplici pacchetti MQTT è essenziale aumentare la dimensione massima del buffer (quella standard è di soli 80 byte).

L'implementazione di tali regole nel nostro sistema IoT non è casuale, bensì risulta da una selezione strategica mirata a verificare l'accuratezza con cui i comandi, ricevuti attraverso il protocollo MQTT, vengano eseguiti sul display di Sense-Hat, assicurandoci così il corretto funzionamento del sistema.

4.3 Software sviluppati

4.3.1 Libreria parse-mqttv5-packet

La libreria parse-mqttv5-packet è uno strumento progettato specificamente per analizzare i pacchetti MQTT, svolge un ruolo fondamentale nell'interpretazione dei dati grezzi, trasformandoli in oggetti strutturati e facilmente manipolabili, facilitando così la loro analisi.

La necessità di una tale libreria è emersa dall'osservazione che le soluzioni esistenti non si adattavano in modo ottimale alle specifiche esigenze del nostro progetto. Pertanto, è stata presa la decisione di sviluppare una libreria dedicata che potesse offrire un'analisi precisa e flessibile dei pacchetti MQTT, fornendo agli sviluppatori un'interfaccia intuitiva e potente per lavorare con i messaggi del protocollo.

La classe Parser è progettata per essere estremamente versatile; essa può essere configurata al momento della creazione fornendo al suo costruttore la versione del protocollo MQTT dei pacchetti da analizzare. Una volta istanziata con la versione specificata, il parser è pronto per eseguire il suo compito.

Il fulcro operativo di Parser risiede nel suo metodo parse, il metodo prende in input un buffer di byte, che può contenere uno o più pacchetti MQTT consecutivi, riconosce il tipo di ogni pacchetto con i contenuti specifici e decifra il campo "lunghezza rimanente" (Figura 33).

Il campo "lunghezza rimanente" nei pacchetti MQTT è cruciale perché indica quanti byte leggere per completare l'analisi del pacchetto in questione. Il metodo parse() utilizza queste informazioni per identificare i confini di ogni pacchetto all'interno del buffer, consentendo al metodo di posizionarsi correttamente nel buffer per iniziare l'analisi del successivo pacchetto dopo aver terminato quella del corrente.

La codifica della "lunghezza rimanente" è implementata in questo modo: ogni byte letto contribuisce al valore totale, con i primi 7 bit di ciascun byte che rappresentano il valore e l'ottavo bit che indica se ci sono altri byte da leggere. Questa informazione è essenziale al parser per determinare con precisione dove un pacchetto termina e inizia

il successivo, permettendo una gestione accurata di flussi di pacchetti concatenati senza errori o ambiguità.

Una volta riconosciuto il tipo di pacchetto, parse() procede con la creazione di un oggetto di tipo Packet, popolandone gli attributi con i dati del pacchetto specifico estratti dal buffer.

La classe Packet, è il cuore della libreria parse-mqttv5-packet, funge da classe base astratta per tutti i diversi tipi di pacchetti MQTTv5. Definisce gli attributi comuni e l'interfaccia che ogni pacchetto ha, stabilendo una struttura condivisa dalle classi figlie; questa gerarchia di inheritance garantisce un'organizzazione modulare e intuitiva e il riutilizzo del codice. Ogni classe figlia di Packet rappresenta un tipo specifico di pacchetto MQTT:

- **Connect:** Tratta e rappresenta i pacchetti di tipo CONNECT, contenenti informazioni per stabilire una connessione tra il client e il server.
- **ConnAck**: Tratta e rappresenta i pacchetti CONNACK, che sono le risposte del server ai pacchetti CONNECT.
- **Publish** Tratta e rappresenta i pacchetti PUBLISH, usati per distribuire messaggi ai client sottoscritti.
- **PubAck**: Tratta e rappresenta i pacchetti PUBACK, inviati come risposta ai pacchetti PUBLISH con QoS 1.
- **PubRec**: Tratta e rappresenta i pacchetti PUBREC, utilizzati nel flusso di QoS 2 per segnalare la ricezione di un pacchetto PUBLISH.
- **PubRel**: Tratta e rappresenta i pacchetti PUBREL, che fanno parte del processo di assicurazione di consegna per i pacchetti con QoS 2.
- **PubComp**: Tratta e rappresenta i pacchetti PUBCOMP, che completano il ciclo di scambio per i pacchetti PUBLISH con QoS 2.
- **Subscribe**: Tratta e rappresenta i pacchetti SUBSCRIBE, usati dai client per richiedere la sottoscrizione a topic specifici.
- **SubAck**: Tratta e rappresenta i pacchetti SUBACK, inviati dal server in risposta ai pacchetti SUBSCRIBE.
- **UnSubscribe**: Tratta e rappresenta i pacchetti UNSUBSCRIBE, utilizzati dai client per annullare le sottoscrizioni.
- **UnSubAck**: Tratta e rappresenta i pacchetti UNSUBACK, che sono le risposte del server agli UNSUBSCRIBE.

Ognuna di queste classi eredita gli attributi della classe Packet e introduce ulteriori dettagli e funzionalità specifiche dipendenti dal tipo di pacchetto che rappresenta.

Ciascuna classe figlia dispone di un costruttore che è responsabile di analizzare il resto del buffer non ancora interpretato, estrarre e popolare i campi specifici del pacchetto, attraverso la definizione di nuove proprietà.

Questo approccio offre il grosso vantaggio agli utilizzatori di questa libreria la realizzazione di funzioni generiche capaci di accettare come input un'istanza di Packet e operare con qualsiasi delle sue sottoclassi grazie al polimorfismo.

In sintesi, questa architettura non solo rende la libreria estremamente flessibile ma permette anche un'interoperabilità polimorfica, consentendo lo sviluppo di funzioni generiche che possono operare su qualsiasi sottoclasse di Packet. Ciò offre agli sviluppatori la possibilità di estendere la libreria per supportare nuovi tipi di pacchetti o per adattarsi a future versioni del protocollo MQTT, mantenendo una gestione efficace e coerente di vari tipi di comunicazioni.

La libreria è pubblicamente accessibile su GitHub, consentendo quindi agli sviluppatori interessati di contribuire, personalizzare e integrare questa soluzione nelle loro applicazioni di analisi. Per ulteriori dettagli, esplorazioni e contributi, è possibile visitare il repository parse-mqttv5-packet su GitHub .

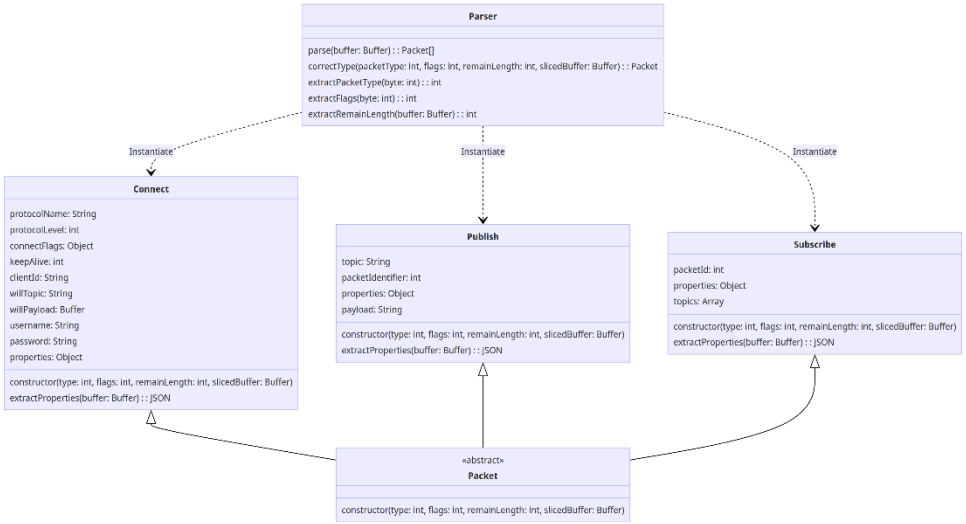

Figura 33. Class diagram parse-mqttv5-packet

4.3.2 Adapter: ponte essenziale tra Falco e il Monitor

Questo paragrafo analizza il processo di progettazione e realizzazione dell'Adapter software, una componente fondamentale ideata per ottimizzare l'integrazione e il flusso di dati tra FALCO e il MONITOR.

Il seguente sequence diagram (Figura 15), illustra gli aspetti architetturali del sistema:

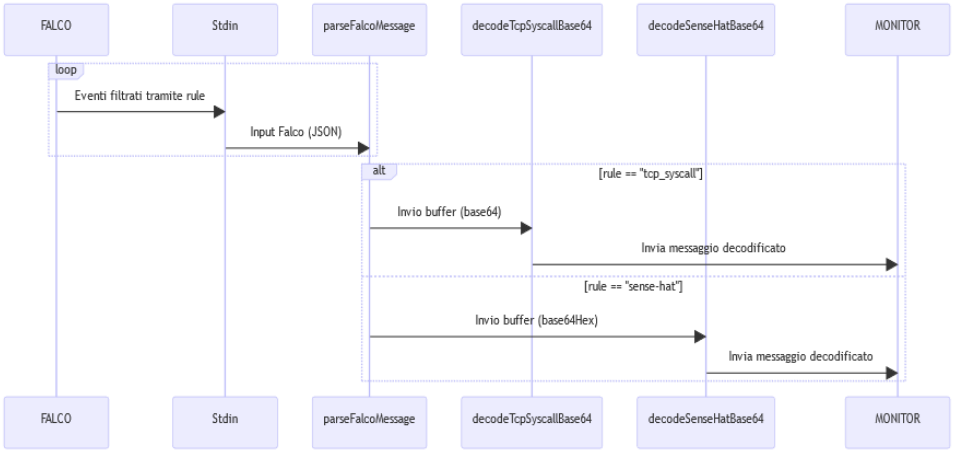

Figura 15. Sequence diagram dell'Adapter (in maiuscolo attori esterni)

Il diagramma illustra con chiarezza il flusso operativo, mostrando come gli eventi, inizialmente filtrati da Falco secondo regole ben definite, vengono inviati mediante una pipe in formato JSON all'Adapter, che li elabora utilizzando la funzione di parsing chiamata parseFalcoJSON.

Un aspetto fondamentale in questa fase è l'analisi del campo 'rule' presente nei dati JSON, che può essere 'tcp_syscall' o 'sense-hat', infatti il parsing non è soltanto un'operazione di interpretazione dei dati, ma identifica anche quale regola specifica di Falco è stata innescata. Questa distinzione è fondamentale perché dirige l'evento alla funzione di decodifica appropriata: decodeTcpSyscallBase64 per 'tcp_syscall' o decodeSenseHatBase64 per 'sense-hat'.

La funzione decodeSenseHatBase64 trasforma i dati in base64 trovati in 'evt.buffer' in una stringa esadecimale che rappresenta un colore. Questo dato viene poi convertito in un oggetto RGB565 standard tramite la funzione hexToRGB565, che isola i componenti rosso, verde e blu.

Sono state implementate due versioni dell'Adapter: una versione light e una full, che differiscono nel modo in cui processano e inviano i dati al monitor, la differenza principale si trova proprio nella funzione decodeTcpSyscallBase64 e nel modo in cui le due versioni trasformano il messaggio RGB888 ricevuto:

La versione full, oltre ad analizzare i messaggi MQTT, li modifica anche, convertendo i messaggi dal formato RGB888 al formato RGB565.

La versione light invece si limita solamente al parsing dei messaggi MQTT mantenendo il formato RGB888. Ciò comporterà un maggiore impegno per il

MONITOR nella fase successiva; infatti, dovrà effettuare delle operazioni supplementari autonomamente per poter confrontare i dati relativi al colore visualizzato sul Sense-Hat con quelli ricevuti dal broker MQTT.

Un'importante sfida nello sviluppo dell'Adapter è stata la gestione dell'asimmetria tra i pacchetti ‘tcp_syscall’ e ‘sense-hat’, quando l'Adapter riceve pacchetti ‘tcp_syscall’, può trovarsi di fronte a una situazione in cui ‘evt.buffer’ contiene più messaggi MQTT concatenati. D'altra parte invece, i pacchetti ‘sense-hat’ non presentano questa complessità poiché sono legati a eventi singoli e non contengono mai messaggi multipli in un singolo ‘evt.buffer’.

L'asimmetria sorge quindi dal fatto che l'Adapter deve gestire due flussi di dati diversi: uno (da ‘tcp_syscall’) potenzialmente composto da più messaggi MQTT e l'altro (da ‘sense-hat’). L'obiettivo è riuscire a garantire che i dati vengano elaborati e inoltrati al monitor in modo sincronizzato e accurato. L'Adapter adotta una coda FIFO (First In, First Out) come soluzione strategica per affrontare la disomogeneità tra i pacchetti ‘tcp_syscall’ e ‘sense-hat’, un aspetto cruciale per il corretto funzionamento del sistema.

La coda viene utilizzata per immagazzinare i messaggi MQTT decodificati dal campo ‘evt.buffer’ nei pacchetti ‘tcp_syscall’, assicurando la sequenzialità nel loro trattamento tramite l'inserimento in una coda dei messaggi MQTT. L'Adapter, quindi, non procede all'immediato inoltro dei messaggi al monitor; piuttosto, si pone in attesa dell'arrivo di un pacchetto ‘sense-hat’ per avviare il processo di elaborazione. Questo approccio sincronizzato assicura che i messaggi MQTT siano trattati in maniera sequenziale con gli eventi ‘sense-hat’, evitando potenziali disallineamenti nei dati elaborati. Alla ricezione di un pacchetto ‘sense-hat’, l'Adapter inizia l'inoltro dei messaggi dalla coda FIFO verso il monitor. Questo processo continua fino all'identificazione di un messaggio MQTT che si allinea o corrisponde con l'evento ‘sense-hat’ in questione. Questo metodo garantisce che i dati siano processati e trasmessi in modo ordinato, rispettando la necessaria sequenzialità e sincronizzazione per un monitoraggio efficace (figura 16).

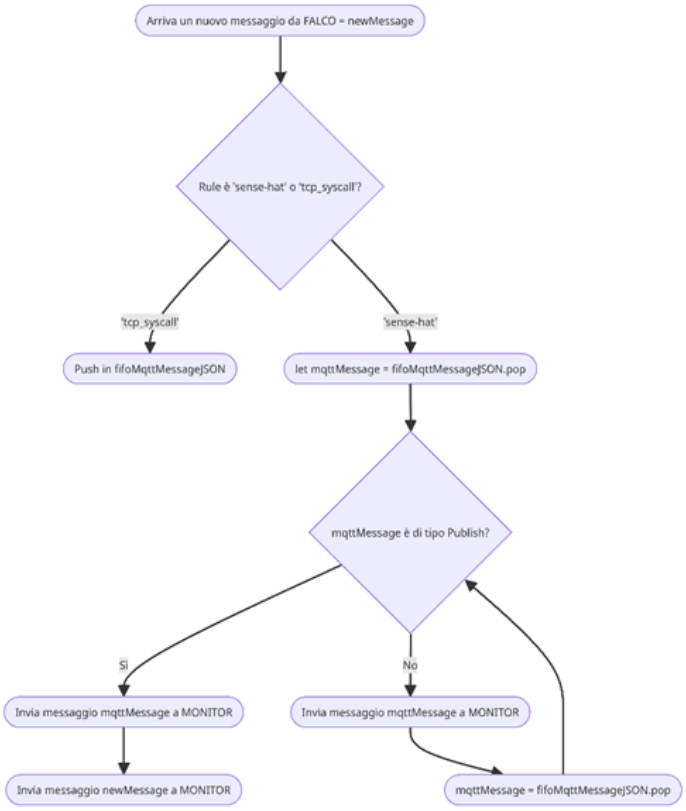

Figura 16. Garanzia della sequenzialità tramite FIFO: Flow Chart

È importante notare che nel caso in cui il codice monitorato dovesse comandare il sense-hat in modo indipendente dai messaggi MQTT, il MONITOR sarebbe in grado di rilevarlo successivamente.

4.3.3 Monitor con specifiche in RML

Il monitor, denominato RMLatDIBRIS Monitor, è una componente cruciale nel nostro sistema, incaricato di osservare e valutare in tempo reale le attività del sistema. Funziona come una sentinella, verificando costantemente che l'esecuzione del programma sia conforme alle specifiche stabilite. Grazie alla sua attività di monitoraggio, è possibile rilevare qualsiasi comportamento anomalo o non conforme durante l'esecuzione del programma, permettendo di intervenire immediatamente (o persino automaticamente) per apportare le necessarie correzioni (RMLatDIBRIS Monitor, 2024).

Essendo implementato in Prolog, richiede l'installazione di SWI-Prolog versione 7 o superiore per funzionare. Inoltre, per definire le regole e le condizioni che gli eventi devono soddisfare affinché il sistema sia considerato conforme, è necessario fornire una specifica come input.

Il monitor può essere utilizzato in due modalità: offline e online, nella modalità offline, il monitor legge una traccia di sequenze da un file, e in base alla specifica definita, genera report sul rispetto o meno delle condizioni specificate, la modalità offline è utile per analizzare sequenze di eventi passati o per testare il comportamento del monitor con dati noti.

Nella modalità online, invece, il monitor riceve gli eventi in tempo reale tramite un'interfaccia TCP, questa modalità è ideale per sistemi che richiedono monitoraggio in tempo reale, permettendo di intercettare e analizzare gli eventi man mano che si verificano (RML Dibris, 2024).

Per definire le specifiche di monitoraggio, si utilizza il linguaggio RML (Runtime Monitoring Language), appositamente progettato per questo scopo. Gli sviluppatori possono definire facilmente regole utilizzando RML per garantire la conformità delle tracce di esecuzione del programma (Ancona, 2021).

Per utilizzare RML, è necessario utilizzare il RMLatDIBRIS Compiler, che trasforma le specifiche RML in codice Prolog.

Per il nostro sistema, sono state sviluppate due specifiche RML distinte che, pur essendo simili presentano differenze cruciali nel meccanismo di confronto tra il messaggio ‘tcp_syscall’ e ‘sense-hat’.

La specifica ‘lightAdapterRule.rml’ utilizzata in combinazione con l'AdapterLight e il monitor è chiamato a svolgere calcoli supplementari per realizzare un controllo incrociato tra ‘sense-hat’ e ‘tcp_syscall’. I messaggi del primo tipo includono dati in formato RGB565, mentre i secondi sono in formato RGB888 (Figura 17). Per stabilire la correlazione tra queste due tipologie di messaggi, il monitor deve verificare che le seguenti equazioni siano soddisfatte: ((red888 / 8) – red565) ≤ 1, ((green888 / 4) – green565) ≤ 1 e ((blue888 / 8) – blue888) ≤ 1.

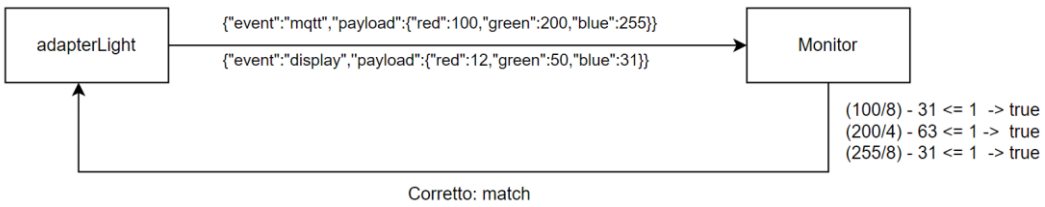

Figura 17. Diagram sul funzionamento del Monitor in caso di match

Al contrario, la specifica "fullAdapterRule.rml" mira a facilitare l'attività del monitor. Poiché l'adapterFull si occupa della conversione da RGB888 a RGB565, il monitor deve solamente verificare l'uguaglianza dei due messaggi.

In questo secondo scenario (Figura 18), l'obiettivo principale è aumentare il carico di lavoro sull'Adapter per alleggerire quello sul monitor.

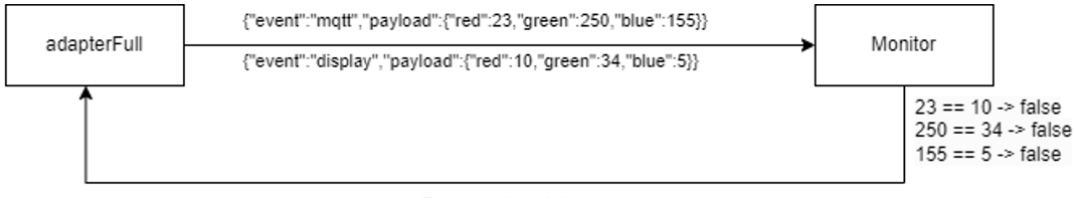

Figura 18. Diagram sul funzionamento del Monitor in caso di non match

4.4 Valutazione delle Prestazioni e Impatto sull'Overhead

L'analisi dell'overhead e dell'impatto sulle performance dell'intero sistema di monitoraggio viene suddivisa in quattro livelli distinti, ognuno dei quali esamina l'impatto incrementale degli strumenti utilizzati sulle prestazioni generali del sistema.

Al **Livello 0**, il sistema viene esaminato nella sua configurazione di base, privo di qualsiasi meccanismo di monitoraggio implementato. Questa condizione iniziale funge da punto di riferimento, consentendo una valutazione comparativa dell'efficacia dei successivi strumenti di monitoraggio. In questa fase, la performance del sistema viene analizzata in termini di utilizzo delle risorse, latenza, fornendo così una baseline per le valutazioni future.

Successivamente, il **Livello 1** introduce Falco. In questa fase, l'attenzione si concentra sull'impatto di Falco sulle prestazioni generali del sistema, valutando le variazioni nell'utilizzo delle risorse, nella latenza e nel throughput, analizzando quindi l'influenza del processo di rilevamento degli eventi.

Il **Livello 2** amplia ulteriormente l'analisi, incorporando uno stack di monitoraggio completo che comprende Falco, AdapterLight e il Monitor. L'obiettivo di questa fase è valutare come l'interazione sinergica tra questi componenti influenzi le prestazioni globali del sistema. Particolare attenzione è rivolta al ruolo dell'Adapter nella gestione dei dati, inclusa la decodifica dei messaggi MQTT e la sincronizzazione dei flussi di dati, al fine di determinare il loro effetto complessivo sulle prestazioni.

Infine, il **Livello 3** replica la configurazione del Livello 2, ma sostituisce AdapterLight con AdapterFull, permettendo così un confronto diretto tra le due configurazioni per determinare l'impatto della versione full rispetto a quella light.

Nel contesto dell'analisi delle prestazioni del sistema, è stato utilizzato uno script Bash appositamente progettato per orchestrare in modo sistematico la raccolta dei dati prestazionali. Lo script è stato concepito per monitorare l'impatto sull'intero sistema derivante dall'esecuzione di uno o più programmi. L'esecuzione dello script richiede due parametri: il numero di iterazioni (<numero_iterazioni>) e la durata di ciascuna iterazione (<durata_iterazione>). Dopo l'avvio, lo script procede con la creazione di una directory dedicata (./livelloi) per l'archiviazione dei file di output generati.

È stata adottata una strategia di monitoraggio frammentata in sessioni distinte per consentire di valutare l'impatto sulle prestazioni dell'intero sistema, garantendo al contempo la capacità di calcolare medie per ogni insieme di iterazioni. Ciò offre una rappresentazione equilibrata e normalizzata delle prestazioni, riducendo il rischio che eventi esterni o fluttuazioni atipiche influenzino i risultati. Una volta avviato il processo 'clientSubSenseHat.js' più lo stack di monitoraggio relativo al livello in analisi, segue l'avvio di due processi di monitoraggio paralleli in ogni iterazione: il primo utilizza 'mpstat' per tracciare l'uso della CPU, mentre il secondo impiega 'perf' per raccogliere statistiche sulle istruzioni eseguite e i cicli di processore.

Al termine delle iterazioni, lo script termina il monitoraggio e indica la conclusione della raccolta dati, indipendentemente dal livello di monitoraggio considerato. Questo approccio metodologico assicura la precisione e la riproducibilità delle misurazioni, elementi cruciali per l'analisi comparativa delle prestazioni in scenari diversificati.

```bash
#!/bin/bash
# Definizione del percorso della cartella di output e creazione della cartella
OUTPUT_DIR="./livello2"
mkdir -p $OUTPUT_DIR
echo "Cartella $OUTPUT_DIR creata per i file di output."
# Avvio di Falco in background
echo "Avvio di Falco pipe adapterLight in background"
sudo falco -c falco/falco.yaml -r falco/log_server.yaml -r falco/sense-hat.yaml -b -A -S 4000 | node ws/adapterLight.js &
FALCO_PID=$!
# Avvio node clientSubSenseHat.js in background
echo "Avvio di node clientSubSenseHat.js in background."
node clientSubSenseHat.js &
NODE_PID=$!
for i in $(seq 1 $ITERATIONS); do
    echo "Inizio iterazione $i."
    # Avvia mpstat in background per registrare l'uso della CPU
    mpstat 1 $DURATION > "$OUTPUT_DIR/mpstat_output_$i.txt" &
    MPSTAT_PID=$!
    # Avvia perf in background per monitorare il sistema
    perf stat -a -e instructions,cycles -o "$OUTPUT_DIR/perf_output_$i.txt" sleep $DURATION &
    PERF_PID=$!
    # Aspetta che il campionamento di DURATION secondi finisca per ogni processo
    wait $MPSTAT_PID
    wait $PERF_PID
done
kill $FALCO_PID
kill $NODE_PID
```

Figura 19. Script bash per la raccolta dei dati di livello 2

4.4.1 Primo caso di test: Scenario realistico

In questo scenario di test, abbiamo esaminato un campione di 10 sessioni, ognuna della durata di 100 secondi, il publisher (clientPub.js) invia un messaggio ogni 2 secondi.

- **Livello 0:** L'analisi ha rivelato che il numero medio di istruzioni elaborate ammontava a 0.8×10^9, mentre il numero medio di cicli di processore a 1.1×10^9, determinando un utilizzo complessivo medio della CPU dello 0.21% (Figura 20).

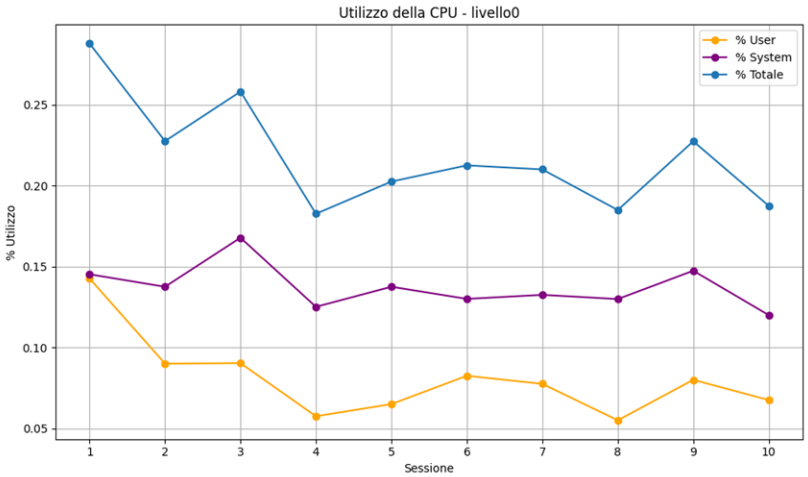

Figura 20. Scenario realistico: utilizzo della CPU nel livello 0

- **Livello 1:**

  La media delle istruzioni elaborate è stata di 1.38×10^9, e la media dei cicli di processore è stata di 1.8×10^9.

  L'utilizzo medio della CPU per User è stato del 0.14% mentre per System del 0,27%, con un utilizzo medio complessivo della CPU del 0.41% (Figura 21).

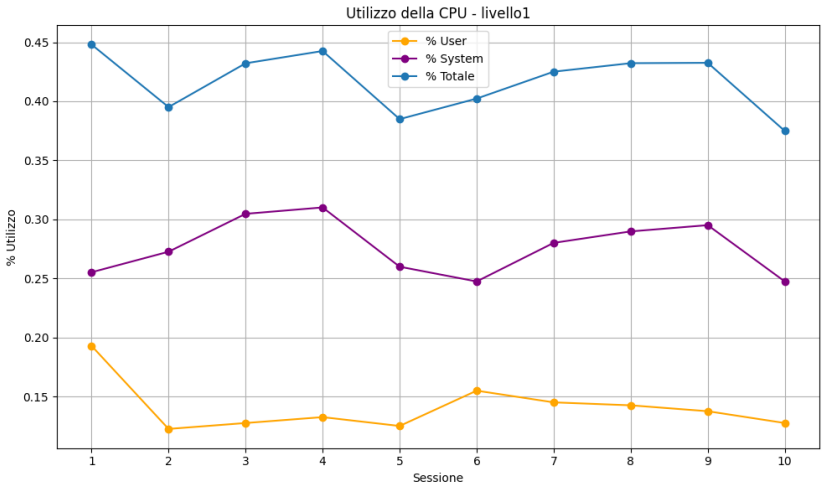

Figura 21. Scenario realistico: utilizzo della CPU nel livello 1

- **Livello 2:** La media delle istruzioni elaborate è stata di 2.06×10^9, e la media dei cicli di processore è stata di 2.82×10^9, evidenziando un ulteriore incremento che sottolinea un aumento continuo della complessità e del carico di lavoro nel sistema.

  L'utilizzo medio della CPU per User è stato del 0.23% mentre per System del 0,41% per una media totale di utilizzo della CPU del 0.64% (Figura 22).

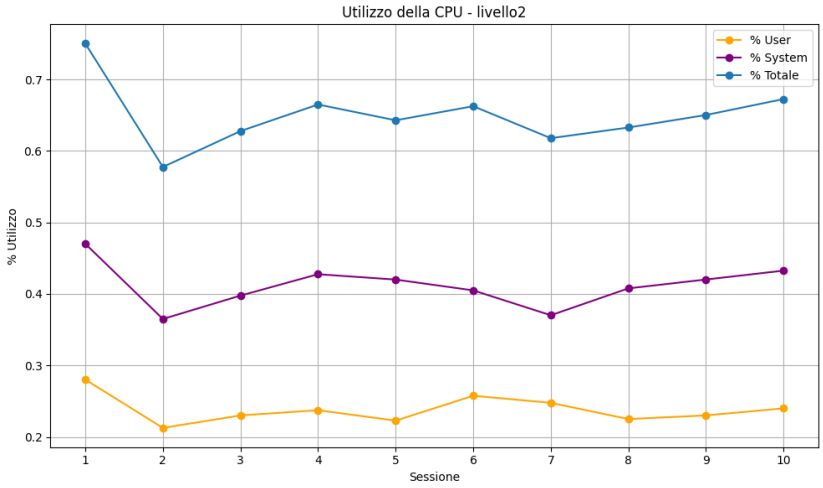

Figura 22. Scenario realistico: utilizzo della CPU nel livello 2

- **Livello 3:** La media delle istruzioni elaborate è stata di 2.04×10^9, e la media dei cicli di processore è stata di 2.80×10^9, evidenziando un ulteriore incremento che sottolinea un aumento continuo della complessità e del carico di lavoro nel sistema. L'utilizzo medio della CPU per User è stato del 0.24% mentre per System del 0.40% per una media totale di utilizzo della CPU del 0.64% (Figura 23).

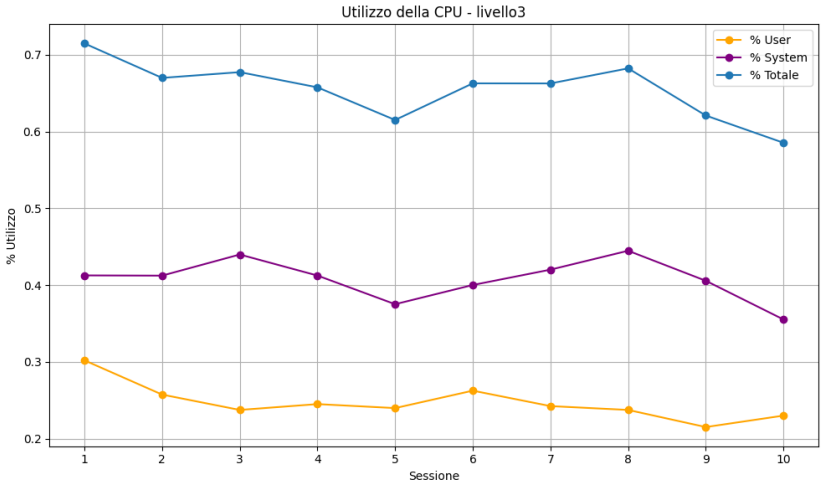

Figura 23. Scenario realistico: utilizzo della CPU nel livello 3

Si osserva un incremento progressivo delle percentuali totali di utilizzo della CPU, denotando un aumento delle iterazioni e un maggiore carico sulla CPU (Figura 24).

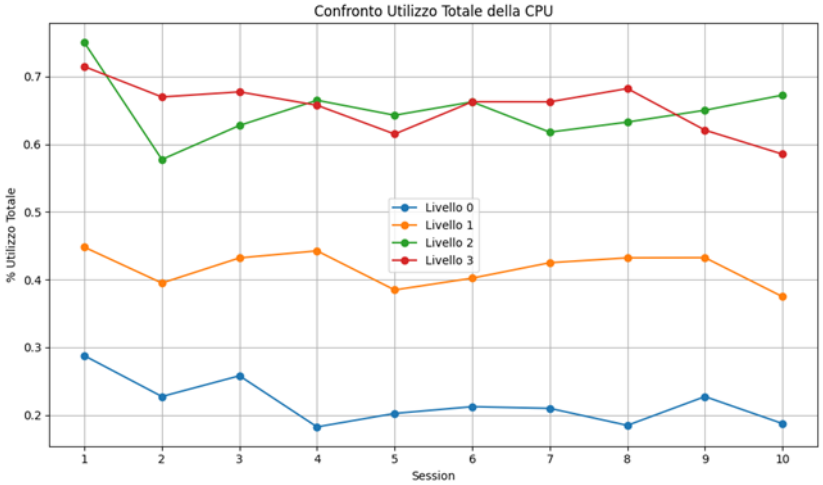

Figura 24. Scenario realistico: confronto utilizzo CPU per livello

Lo stesso si è notato nelle istruzioni elaborate e nei cicli di clock (Figura 25-26):

- Il passaggio dal Livello 0 al Livello 1 mostra un aumento percentuale delle istruzioni di circa 58.30% e dei cicli di circa 62.93%%.
- Il passaggio dal Livello 1 al Livello 2 mostra un aumento percentuale delle istruzioni di circa 62.93% e dei cicli di circa 51.52%.
- Il passaggio dal Livello 2 al Livello 3 mostra un decremento percentuale delle istruzioni di circa 0.84% e dei cicli di circa 0.74%.

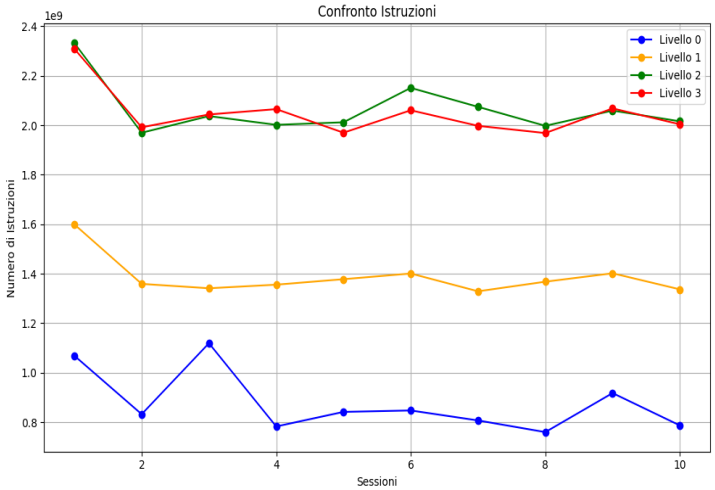

Figura 25. Scenario realistico: confronto istruzioni eseguite per livello

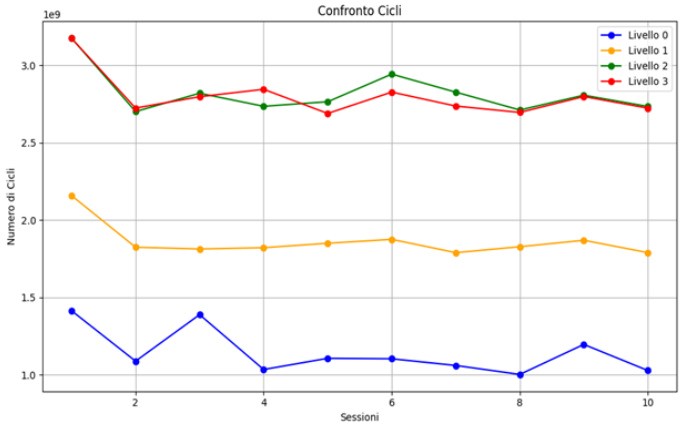

Figura 26. Scenario realistico: confronto cicli di clock per livello

Istruzioni eseguite: Il primo grafico mostra il numero di istruzioni eseguite in ogni test per i tre livelli. Si osserva che con l'aumentare del livello, il numero di istruzioni tende ad aumentare. Questo può essere attribuito alla complessità aggiuntiva o alle funzionalità implementate nei livelli superiori.

Cicli di clock: Analogamente, il secondo grafico evidenzia il numero di cicli di clock per ogni test, mostrando anch'esso un incremento man mano che si passa da un livello all'altro. L'aumento dei cicli riflette il maggior carico di lavoro e la potenziale complessità incrementata nei livelli superiori.

4.4.2 Secondo caso di test: Stress test

In questo scenario di test, abbiamo esaminato un campione di 10 sessioni, ognuna della durata di 100 secondi, clientPub.js invia un messaggio ogni 100 millisecondi.

- **Livello 0:** L'analisi ha rivelato che il numero medio di istruzioni elaborate ammontava a 2.8×10^9, mentre il numero medio di cicli di processore a 4.1×10^9, determinando un utilizzo medio della CPU dello 0,87% (Figura 27).

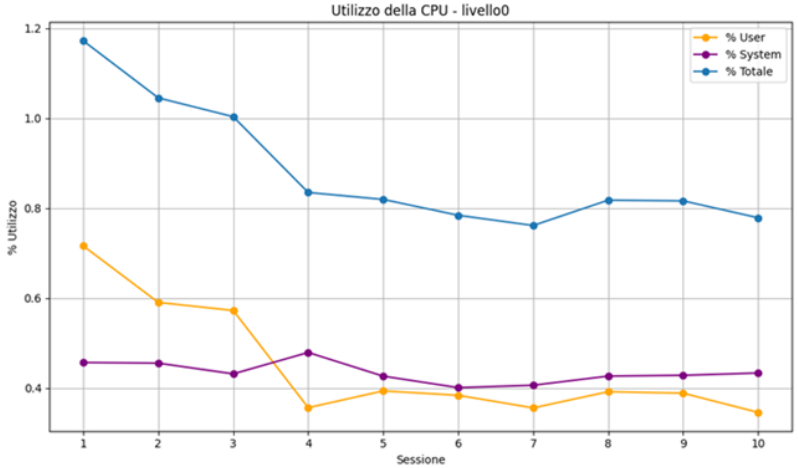

Figura 27. Stress test: utilizzo della CPU nel livello 0

- **Livello 1:** La media delle istruzioni elaborate è stata di 6.2×10^9, e la media dei cicli di processore è stata di 9.2×10^9. Questi risultati indicano un incremento rispetto al Livello 0, riflettendo un aumento del carico di lavoro e della complessità nel sistema.

  L'utilizzo medio della CPU per User è stato del 1.12% mentre per System del 1.26%, con un utilizzo medio complessivo della CPU del 2.38% (Figura 28).

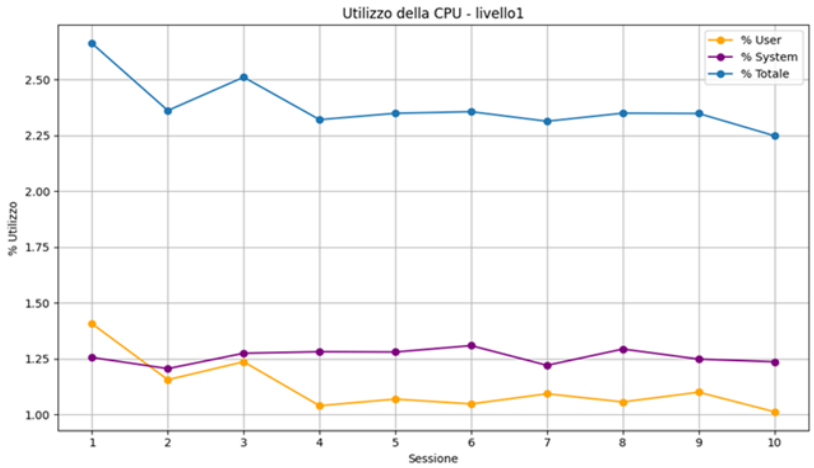

Figura 28. Stress test: utilizzo della CPU nel livello 1

- **Livello 2:** La media delle istruzioni elaborate è stata di 8,0×10^9, e la media dei cicli di processore è stata di 11,8×10^9, evidenziando un ulteriore incremento che sottolinea un aumento continuo della complessità e del carico di lavoro nel sistema. L'utilizzo medio della CPU per User è stato del 1.39% mentre per System del 1.72% per una media totale del 3.11% (Figura 29).

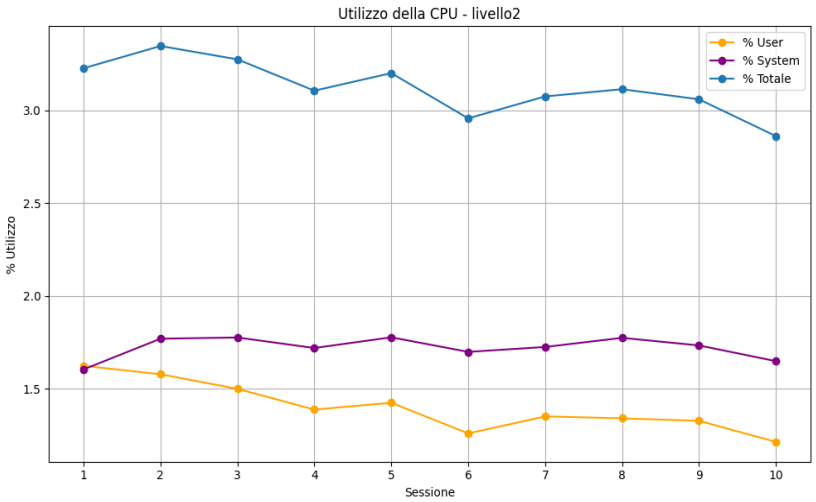

Figura 29. Stress test: utilizzo della CPU nel livello 2

- **Livello 3:** i risultati vengono omessi in quanto, similmente a quanto osservato per il test precedente, si collocano sulla medesima media del livello antecedente e, pertanto, risultano essere poco significativi.

Si è quindi osservato un incremento progressivo delle percentuali totali di utilizzo della CPU, denotando un maggiore carico sulla CPU (Figura 30).

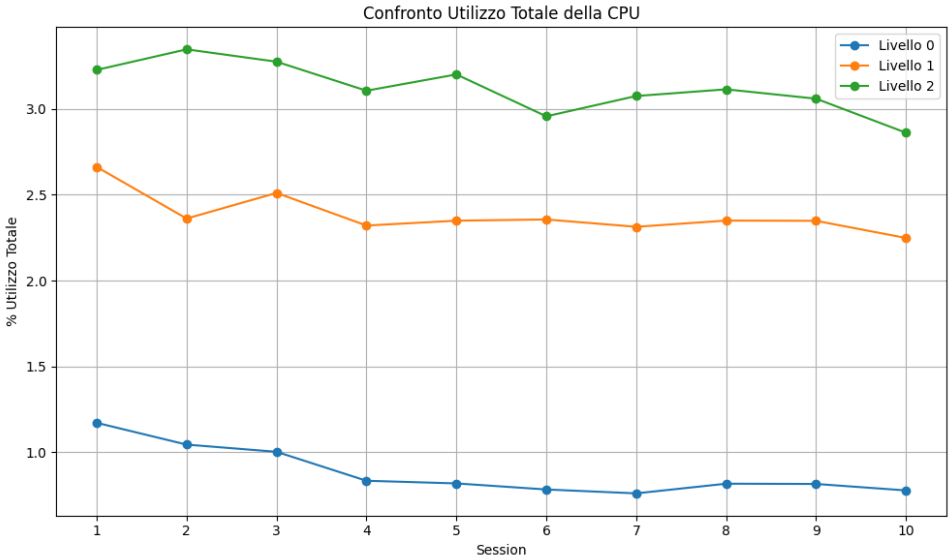

Figura 30. Stress test: confronto utilizzo CPU per livello

Lo stesso si è notato nelle istruzioni elaborate e nei cicli di clock (Figura 31-32):

- Il passaggio dal Livello 0 al Livello 1 mostra un aumento percentuale delle istruzioni di circa 116.60% e dei cicli di circa 123.61%
- l passaggio dal Livello 1 al Livello 2 mostra un aumento percentuale delle istruzioni di circa 28.06% e dei cicli di circa 28.59%.

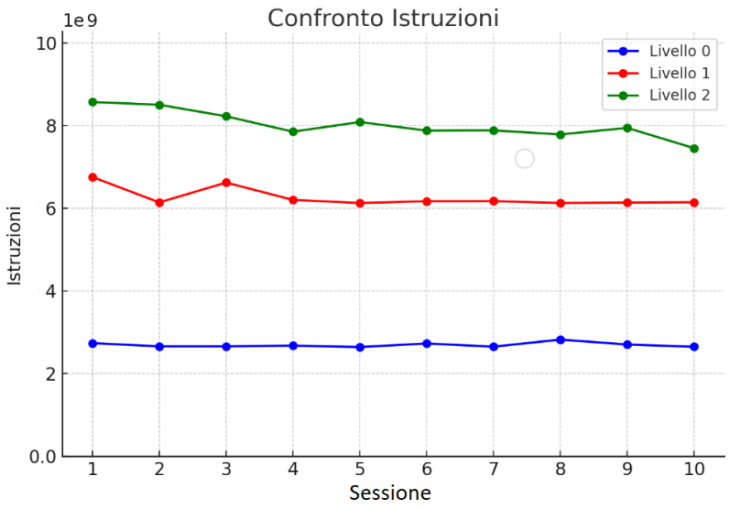

Figura 31. Stress test: confronto istruzioni eseguite per livello

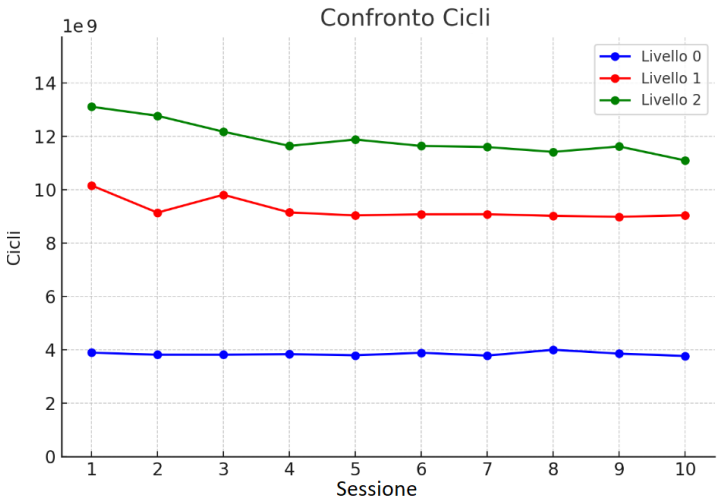

Figura 32. Stress test: confronto cicli di clock per livello

Istruzioni eseguite: Il primo grafico mostra il numero di istruzioni eseguite in ogni test per i tre livelli. Si osserva che con l'aumentare del livello, il numero di istruzioni tende ad aumentare. Questo può essere attribuito alla complessità aggiuntiva o alle funzionalità implementate nei livelli superiori.

Cicli di clock: Analogamente, il secondo grafico evidenzia il numero di cicli di clock per ogni test, mostrando anch'esso un incremento man mano che si passa da un livello all'altro. L'aumento dei cicli riflette il maggior carico di lavoro e la potenziale complessità incrementata nei livelli superiori.

CONCLUSIONI

Questo progetto apre la strada a numerosi sviluppi futuri nel campo del monitoraggio in tempo reale dei dispositivi IoT. La flessibilità dell'approccio proposto permette di esplorare ulteriormente il campo con studi che si basano su dispositivi con capacità computazionali limitate, come i Raspberry Pi.

Finora, il nostro lavoro si è concentrato sui pacchetti MQTT di tipo "publish". Tuttavia, l'utilizzo del Quality of Service (QoS) livello 2 introduce una varietà significativa di pacchetti (come PUBREL e PUBACK) che possono essere sfruttati per eseguire controlli incrociati più sofisticati. Questo non solo amplia l'ambito dell'analisi ma anche aumenta la precisione del monitoraggio, consentendo una valutazione più accurata delle comunicazioni tra dispositivi.

L'introduzione di controlli più rigorosi per analizzare l'overhead generato da questi pacchetti offre una prospettiva interessante. Esaminando come l'overhead varia con l'aumento della complessità dei controlli, possiamo ottenere utili suggerimenti sull’efficienza e sulla scalabilità delle soluzioni di monitoraggio. Un confronto tra gli strumenti di monitoraggio, come l'attuale kmod di Falco e le alternative basate su eBPF e modern BPF, può rivelare differenze significative in termini di prestazioni e capacità di analisi. Quest'analisi comparativa potrebbe guidare lo sviluppo di soluzioni più efficaci e meno invasive. La creazione di un plugin Falco dedicato al parsing dei pacchetti MQTT rappresenta un'altra direzione promettente. Questo permetterebbe di processare i dati direttamente alla fonte, potenzialmente riducendo la latenza e migliorando l'efficienza del sistema di monitoraggio. Studiare la possibilità di applicare l'instrumentazione non solo ai subscriber ma anche ai broker MQTT potrebbe offrire vantaggi in termini di prestazioni.

È possibile sviluppare una versione innovativa dell'Adapter, ottimizzata per ridurre ulteriormente l'overhead. Questo potrebbe essere conseguito attraverso l'adozione di un modello produttore-consumatore implementato su una coda, perfezionando così l'efficienza del sistema e minimizzando la propensione a intoppi. Concludendo, le potenzialità di evoluzione per questo progetto sono numerose e diversificate. Esplorando queste vie, possiamo non solo estendere la nostra comprensione del monitoraggio dei dispositivi IoT in tempo reale ma anche migliorare significativamente l'efficacia e l'efficienza delle soluzioni esistenti.

Bibliografia

1. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti. «Control-flow integrity». In: Proceedings of the 12th ACM conference on Computer and communications security. ACM. 2005, pp. 340–353.
2. Aldweesh, A.; Derhab, A.; Emam, A.Z. Deep learning approaches for anomaly-based intrusion detection systems: A survey, taxonomy, and open issues. Knowl.-Based Syst. 2020, 189, 105124.
3. Al-Fuqaha, A.; Guizani, M.; Mohammadi, M.; Aledhari, M.; Ayyash, M. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. IEEE Commun. Surv. Tutor. 2015, 17, 2347–2376.
4. Ali, Zainab H., Hesham A. Ali, and Mahmoud M. Badawy. "Internet of Things (IoT): definitions, challenges and recent research directions." International Journal of Computer Applications 128.1 (2015): 37-47.
5. Amarlingam M, P. K. Mishra, K. V. V. D. Prasad, and P. Rajalakshmi. Compressed sensing for different sensors: A real scenario for WSN and IoT. In 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT), pp. 289–294, 2016.
6. Ancona D, Luca Franceschini, Angelo Ferrando, Viviana Mascardi, RML: Theory and practice of a domain specific language for runtime verification, Science of Computer Programming, Volume 205, 2021
7. Andrea, I.; Chrysostomou, C.; Hadjichristofi, G. Internet of Things: Security vulnerabilities and challenges. In Proceedings of the 2015 IEEE Symposium on Computers and Communication (ISCC), Larnaca, Cyprus, 6–9 July 2015; pp. 180–187
8. Andy, Syaiful & Rahardjo, Budi & Hanindhito, Bagus. (2017). Attack scenarios and security analysis of MQTT communication protocol in IoT system.
9. AR. Bernat and BP. Miller. «Anywhere, any-time binary instrumentation». In: Proceedings of the 10th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools. 2011, pp. 9–16.
10. Atmoko, Rachmad & Riantini, Rona & Hasin, M. (2017). IoT real time data acquisition using MQTT protocol. Journal of Physics: Conference Series. 853. 012003.
11. Bach, M. Charney, R. Cohn, E. Demikhovsky, T. Devor, K. Hazelwood, A. Jaleel, C. Luk, G. Lyons, H. Patil, et al. «Analyzing parallel programs with pin».

In: Computer 43.3 (2010), pp. 34–41

12. Beecks C, A. Grass, and S. Devasya. Metric Indexing for Efficient Data Access in the Internet of Things. In 2018 IEEE International Conference on Big Data (Big Data), pp 5132–5136, Dec. 2018.
13. Brignon and L. Pierre, "Assertion-Based Verification through Binary Instrumentation," 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), Florence, Italy, 2019, pp. 988-991,
14. Chaabouni, N.; Mosbah, M.; Zemmari, A.; Sauvignac, C.; Faruki, P. Network Intrusion Detection for IoT Security Based on Learning Techniques. IEEE Commun. Surv. Tutor. 2019, 21, 2671–2701
15. Colitti W, Steenhaut K, De Caro N 2011 Integrating Wireless Sensor Networks with the Web Proc. IP+SN Chicago, USA
16. Dhall R. and Vijender Solanki. “An IoT Based Predictive Connected Car Mainte-nance”. In: International Journal of Interactive Multimedia & Artificial Intelligence 4.3 2017.
17. Dıaz M, Cristian Martın, and Bartolomé Rubio. “State-of-the-art, Challenges, and Open Issues in the Integration of Internet of Things and Cloud Computing”. In: Journal of Network and Computer applications, pp. 99–117, 2016.
18. Dorsemaine B., et al. “Internet of Things: A Definition & Taxonomy”. In: 2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies. IEEE, pp. 72–77, 2015.
19. Du, Kunping & Kang, Fei & Shu, Hui & Dai, Li. (2012). Dynamic Binary Instrumentation Technology Overview.
20. Ghayvat H. et al. “WSN and IoT-based Smart Homes and their Extension to Smart Buildings”. In: Sensors, pp. 10350–10379, 2015.
21. Haidhar Athir Mohd Puat and Nor Azlina Abd Rahman 2020 J. Phys.: Conf. Ser. 1712 012009
22. Hassanalieragh M. et al. “Health Monitoring and Management Using Internet-of- Things (IoT) Sensing with Cloud-based Processing: Opportunities and Challenges”. In: 2015 IEEE International Conference on Services Computing, pp. 285–292, 2015.
23. Jin J. et al. “An Information Framework for Creating a Smart City Through Internet of Things”. In: IEEE Internet of Things Journal, pp. 112–121, 2014.
24. Kortuem G., et al. “Smart Objects as Building Blocks for the Internet of Things”.

In: IEEE Internet Computing, pp. 44–51, 2009.

25. Lampkin V et al. 2012 Building smarter planet solutions with MQTT and IBM WebSphere MQ telemetry IBM, ITSO
26. Laurenzano, MM. Tikir, L. Carrington, and A. Snavely. «Pebil: Efficient static binary instrumentation for linux». In: 2010 IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS). IEEE. 2010, pp. 175–183.
27. Mektoubi, H. L. Hassani, H. Belhadaoui, M. Rifi and A. Zakari, "New approach for securing communication over MQTT protocol A comparison between RSA and Elliptic Curve," 2016 Third International Conference on Systems of Collaboration (SysCo), Casablanca, 2016, pp. 1-6.
28. Minerva R, Abyi Biru, and Domenico Rotondi. “Towards a Definition of the Internet of Things (IoT)”. In: IEEE Internet Initiative, pp. 1–86, 2015.
29. Mingwei Zhang, Rui Qiao, Niranjan Hasabnis, and R. Sekar. «A Platform for Secure Static Binary Instrumentation». In: SIGPLAN Not. 49.7 (2014), 129–140. issn: 0362- 1340.
30. Mishra and A. Kertesz, "The Use of MQTT in M2M and IoT Systems: A Survey," in IEEE Access, vol. 8, pp. 201071-201086, 2020,
31. Mubeen S. A. Asadollah, A. V. Papadopoulos, M. Ashjaei, H. Pei-Breivold, and M. Behnam. Management of Service Level Agreements for Cloud Services in IoT: A Systematic Mapping Study. IEEE, pp. 30184–30207, 2018.
32. Russell, B.; Van Duren, D. Practical Internet of Things Security; Packt Publishing Ltd: Birmingham, UK, 2016
33. Taherdoost, H. Security and Internet of Things: Benefits, Challenges, and Future Perspectives. Electronics 2023, 12, 1901.
34. Vermesan O., et al. “Internet of Things Strategic Research Roadmap”. In: Internet of Things-Global Technological and Societal Trends, pp. 9–52, 2011.
35. Vuppalapati, Chandrasekar. Building Enterprise IoT Applications. CRC Press, 2019.
36. Wenzl, G. Merzdovnik, J. Ullrich, and E. Weippl. «From hack to elaborate technique— a survey on binary rewriting». In: ACM Computing Surveys (CSUR) 52.3 (2019), pp. 1–37.
37. Zhang, Mingwei & Qiao, Rui & Hasabnis, Niranjan & Sekar, R.. (2014). A Platform for Secure Static Binary Instrumentation. ACM SIGPLAN Notices.

Sitografia

Blogs, Malware Analysis with Dynamic Binary Instrumentation Frameworks, 2021. [accesso online 04/03/2024]

https://blogs.blackberry.com/en/2021/04/malware-analysis-with-dynamic-binary-instrumentation-frameworks

Falco, The Falco Project, 2024. [accesso online 07/03/2024]

https://falco.org/docs/

RMLatDIBRIS Monitor, 2024. [accesso online 12/03/2024]

https://github.com/RMLatDIBRIS/monitor

RML Dibris, 2024. [accesso online 12/03/2024]

https://rmlatdibris.github.io/

RGB565 Color Picker, 2024. [accesso online 16/03/2024]

https://rgbcolorpicker.com/565

eBPF vs. KMod Performance. [accesso online 16/03/2024]

https://github.com/falcosecurity/libs/issues/267