UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Dipartimento di Scienze Fisiche, Informatiche e Matematiche

Corso di Laurea Magistrale in Informatica (LM-18)

Zero trust e autenticazione per architetture OT industriali

Relatore:
Prof. Mirco Marchetti

Candidato:
Emilio Vecchi
Matricola 192183

ANNO ACCADEMICO 2024/2025

Indice

1 Introduzione **1**

2 Stato dell'Arte **3**

2.1 Tecnologie Industria 4.0 3

2.2 Standard e linee guida per la sicurezza 4

2.2.1 IEC 62443 4

2.2.2 NIST SP 800-82 6

2.3 OpenZiti 11

2.3.1 Paradigma Zero Trust 11

2.3.2 Architettura 13

2.4 WebAuthn 18

2.5 Problemi 22

2.5.1 Autenticazione 22

2.5.2 Modelli di rete 24

3 Analisi di una Linea Pilota **26**

3.1 Descrizione della Linea 26

3.2 Problemi 27

3.2.1 Scansione della rete 27

3.2.2 Intercettazione del traffico 29

3.2.3 Credenziali non sicure 31

4 Progettazione **32**

4.1 Obiettivi 32

4.2 Architettura di Base 33

4.3 Architettura OpenZiti 35

4.4 Autenticazione WebAuthn 40

4.4.1 Database 41

4.4.2 UserHandler: gestione degli utenti . . . 42
4.4.3 LoginGateway: controllo degli accessi . . . 43
4.5 Integrazione su architettura complessa . . . 44
4.5.1 Reverse Proxy . . . 44
4.5.2 Integrazione WebAuthn e OpenZiti . . . 46
4.5.3 Flusso di accesso alle risorse . . . 47

5 Implementazione **50**
5.1 Comunicazione tra i componenti OT . . . 50
5.2 Tecnologie utilizzate . . . 51
5.3 Testbed Virtuale . . . 54
5.3.1 Architettura ICS . . . 54
5.3.2 Integrazione di OpenZiti e WebAuthn . . . 70
5.4 Testbed Fisico . . . 93
5.4.1 Architettura ICS . . . 94
5.4.2 Integrazione di OpenZiti e WebAuthn . . . 102

6 Validazione Sperimentale **109**
6.1 Scenario non sicuro . . . 109
6.1.1 Scansione della rete . . . 109
6.1.2 Intercettazione del traffico . . . 111
6.1.3 Interazione non autorizzata con il PLC . . . 112
6.1.4 Accesso ai servizi a causa di credenziali non sicure . . . 115
6.2 Scenario sicuro . . . 115
6.2.1 Scansione della rete . . . 115
6.2.2 Intercettazione del traffico . . . 116
6.2.3 Interazione non autorizzata con il PLC . . . 117
6.2.4 Accesso ai servizi . . . 118

7 Conclusioni **120**

Riferimenti **122**

Elenco delle Figure

2.1 Accesso alle risorse nel modello Zero [14, Figura 1] 12
2.2 Rete overlay OpenZiti [6] 13
2.3 Diagramma di rete OpenZiti con tunneler [6] 15
2.4 Progetto FIDO2 [2] 18
2.5 Flusso di registrazione WebAuthn [5] 20
2.6 Flusso di autenticazione WebAuthn [5] 21

3.1 Intercettazione comunicazioni tra Controller ABB e dispositivo robotico 30
3.2 Intercettazione comunicazioni tra PLC e Controller ABB 31

4.1 Architettura di base del sistema industriale 35
4.2 Architettura Zero Trust basata su OpenZiti 38
4.3 Flusso di accesso a un servizio nell'overlay OpenZiti 39
4.4 Schema Entity-Relationship del database 42
4.5 Diagramma dei casi d'uso del componente *UserHandler* 43
4.6 Diagramma architettura reverse proxy + *LoginGateway* 45
4.7 Diagramma sequenza reverse proxy + *LoginGateway* 46
4.8 Architettura OpenZiti + WebAuthn 48

5.1 Programma Ladder Logic implementato nel PLC virtuale 56
5.2 Mappatura delle variabili del PLC 57
5.3 Configurazione di un data source Modbus in ScadaBR 65
5.4 Configurazione dei data point associati a un PLC 65
5.5 Dashboard di supervisione della linea 66
5.6 Pagina di login 82
5.7 Codice QR per l'autenticazione 84
5.8 Primo accesso a UserHandler 87
5.9 Pagina principale dello UserHandler 87

6.1 Esempio di traffico catturato sul PLC 111
6.2 Stato del sistema prima dell'interazione 113
6.3 Scrittura delle variabili mediante un client Python 113
6.4 Stato del sistema successivo all'interazione 114
6.5 Esempio di traffico catturato sul PLC nella configurazione sicura 117
6.6 Tentativo di interazione diretta con il PLC mediante client Python nella configurazione sicura 118

Elenco dei Codici

5.1 Dockerfile utilizzato per la costruzione del container OpenPLC 58
5.2 Configurazione automatizzata di OpenPLC 59
5.3 import.del 67
5.4 script enroll.sh 75
5.5 creazione router e identità dei tunneler HIL in bootstrap_tunneler_client.sh 77
5.6 creazione configurazioni host e intercept, servizio e service policy dei tunneler PLC in bootstrap_tunneler_plc.sh 78
5.7 creazione service policy dei tunneler HIL in bootstrap_tunneler_client.sh 79
5.8 configurazione del servizio database tramite OpenZiti 80
5.9 Programma PLC in ST del testbed fisico 95
5.10 Comando di movimento di un singolo giunto (spalla) 98
5.11 Ciclo di controllo basato su lettura dei coil Modbus 98
5.12 Classificazione del colore e aggiornamento del PLC 99
5.13 Controllo del nastro sul Raspberry Pi 4 tramite Modbus e seriale 100
5.14 Gestione dei comandi sul Raspberry Pi Pico 101
5.15 Regole di firewall per limitare l'accesso ai servizi PLC, impostate tramite Ansible . 107

1. Introduzione

La progressiva integrazione tra sistemi *Operational Technology* (OT) e *Information Technology* (IT), favorita dai paradigmi dell'Industria 4.0, ha trasformato profondamente il modo in cui gli impianti industriali vengono progettati, monitorati e gestiti. L'introduzione di dispositivi interconnessi, servizi di supervisione remota, raccolta continua di dati di processo e architetture distribuite ha ampliato le possibilità di automazione e ottimizzazione, ma ha al tempo stesso incrementato la superficie di attacco dei sistemi industriali. In questo contesto, modelli di sicurezza tradizionalmente fondati sulla separazione perimetrale della rete e sulla fiducia implicita nei nodi interni mostrano limiti sempre più evidenti, soprattutto quando l'accesso ai servizi di controllo e supervisione avviene tramite protocolli esposti, credenziali deboli o interfacce direttamente raggiungibili dalla rete locale. A differenza dei sistemi puramente IT, i sistemi di automazione industriale sono caratterizzati da forti vincoli di disponibilità, continuità operativa e sicurezza funzionale. Un accesso non autorizzato a un controllore logico programmabile, a un'interfaccia HMI o a un sistema SCADA non comporta soltanto una violazione della riservatezza delle informazioni, ma può tradursi in un impatto diretto sul comportamento fisico del processo controllato. Per questa ragione, la protezione delle architetture industriali richiede approcci capaci di superare la sola logica perimetrale e di introdurre meccanismi di controllo dell'accesso più granulari, espliciti e coerenti con la natura distribuita degli impianti moderni.

Tra i modelli che rispondono a questa esigenza, il paradigma *Zero Trust* assume un ruolo di particolare interesse. Tale approccio si fonda sul principio secondo cui nessun soggetto, dispositivo o servizio debba essere considerato affidabile per il solo fatto di appartenere a una determinata rete o zona infrastrutturale. L'accesso alle risorse deve invece essere mediato da verifiche esplicite di identità, autorizzazione e contesto, riducendo la fiducia implicita tipica delle architetture tradizionali. In parallelo, l'adozione di meccanismi di autenticazione forte e passwordless, come quelli basati su *WebAuthn*, consente di rafforzare ulteriormente la protezione delle interfacce applicative, superando alcune delle debolezze più comuni legate all'uso di password statiche, riutilizzate o predefinite.

All'interno di questo quadro, la presente tesi analizza l'applicazione congiunta di un modello *Zero Trust* e di un sistema di autenticazione passwordless basato su *WebAuthn* in un contesto OT.

L'obiettivo del lavoro è progettare, implementare e valutare una soluzione in grado di limitare l'esposizione diretta dei servizi industriali, mediare l'accesso alle interfacce di controllo e supervisione e impedire interazioni non autorizzate con i componenti del processo. A tal fine, l'analisi non si limita a una trattazione teorica delle tecnologie impiegate, ma si sviluppa attraverso un percorso che comprende lo studio di una linea pilota reale, la definizione di requisiti di sicurezza coerenti con le criticità osservate, la progettazione dell'architettura protetta e la sua implementazione su testbed sperimentali.

2. Stato dell'Arte

Questo capitolo presenta lo stato dell'arte relativo alle tecnologie e agli approcci alla sicurezza applicabili alle architetture OT industriali. Verranno analizzati i principali standard di riferimento, oltre ai modelli e alle architetture emergenti, come lo Zero Trust e i meccanismi di autenticazione passwordless.

2.1 Tecnologie Industria 4.0

Negli ultimi anni il settore industriale ha attraversato un radicale processo di trasformazione guidato dall'adozione del paradigma di Industria 4.0. Uno degli elementi chiave di questo modello è l'interconnessione tra sistemi OT (Operational Technology) e IT (Information Technology), integrando quindi infrastrutture di comunicazione, piattaforme per la raccolta e analisi di dati e software per la gestione dei processi industriali in modo efficiente e automatizzato.

Tra le principali tecnologie introdotte si possono individuare:

- **Cyber-Physical Systems (CPS)**: componenti meccanici controllati e monitorati da algoritmi informatici, integrati con la rete e con gli utenti. Grazie all'unione di elementi fisici e software, sono in grado di operare in diverse modalità ed interagire tra loro in molteplici modi in base al contesto in cui si trovano.
- **Internet of Things (IoT)**: paradigma in cui oggetti fisici dotati di sensori, attuatori, capacità di elaborazione e connettività di rete sono in grado di raccogliere dati, elaborare informazioni e interagire con l'ambiente o con altri sistemi.
- **Cloud Computing**: modello di erogazione di risorse informatiche tramite Internet, accessibili su richiesta, fornite dinamicamente, senza la necessità di gestire direttamente l'infrastruttura fisica sottostante.
- **Artificial Intelligence**: insieme di tecniche che permettono a sistemi computazionali di apprendere dai dati, riconoscere pattern e supportare processi decisionali.

Viene inoltre applicato il concetto di smart factory, un ambiente di produzione altamente digitalizzato e connesso, che si articola in tre componenti principali:

- **Smart Production**: tecnologie che favoriscono la collaborazione tra operatori, macchine e strumenti.
- **Smart Service**: servizi digitali basati sui dati generati dai sistemi industriali interconnessi, che consentono funzionalità di monitoraggio, analisi e manutenzione.
- **Smart Energy**: tecnologie orientate alla sostenibilità, che permettono di ottimizzare l'utilizzo delle risorse e migliorare l'efficienza energetica.

Come risultato la manodopera umana viene potenziata, eliminando lo svolgimento di funzioni ripetitive e le macchine sono inoltre in grado di prendersi in carico compiti sempre più complessi, richiedendo interventi manuali solo in caso di eccezioni o imprevisti. Tuttavia, l'elevato numero di servizi integrati nelle infrastrutture di rete introdotti dalle tecnologie dell'Industria 4.0, comporta un significativo ampliamento della superficie di attacco dei sistemi industriali. L'esposizione ad un numero crescente di minacce informatiche, ha reso necessario adottare modelli di sicurezza più avanzati, in grado di mitigare le vulnerabilità ereditate dagli approcci tradizionali basati sull'isolamento delle reti, oggi non più sufficienti.

2.2 Standard e linee guida per la sicurezza

La crescente integrazione tra sistemi IT e OT e il conseguente aumento della connettività nei sistemi industriali hanno reso necessario lo sviluppo di standard e linee guida per la sicurezza in questo ambito. A differenza delle reti tradizionali, sono richiesti requisiti specifici in termini di affidabilità, disponibilità e sicurezza operativa, rendendo necessaria l'adozione di modelli di sicurezza dedicati. In questa sezione verranno analizzati i principali standard sviluppati dagli enti di riferimento del settore della sicurezza informatica in ambito industriale.

2.2.1 IEC 62443

La serie di standard IEC 62443 è stata sviluppata dalla International Electrotechnical Commission (IEC) in collaborazione con la International Society of Automation (ISA). Essa definisce un fra-

mework completo che determina linee guida per la progettazione, l'implementazione e la gestione della sicurezza lungo l'intero ciclo di vita dei sistemi. In particolare, assumono un ruolo centrale requisiti quali la disponibilità dei sistemi, la sicurezza operativa e la continuità dei processi produttivi, che devono essere garantiti anche in presenza di possibili minacce informatiche.

La serie di standard è organizzata in sei aree:

1. **General**: definizioni, terminologia e modelli concettuali;
2. **Policies and Procedures**: linee guida per la gestione degli impianti da parte degli operatori;
3. **System**: requisiti per la progettazione e l'integrazione dei sistemi industriali;
4. **Components and Requirements**: requisiti per i singoli componenti dei sistemi di automazione;
5. **Profiles**: profili pensati per definire requisiti applicativi o settoriali specifici;
6. **Evaluation**: metodologie di valutazione della sicurezza, per garantire che i risultati delle verifiche siano coerenti e riproducibili.

Tra i diversi aspetti introdotti dalla serie IEC 62443, particolare rilevanza per il presente lavoro è assunta dal concetto di *Security Level* (SL), in quanto strettamente legato alla definizione dei requisiti di sicurezza e alla gestione del rischio nei sistemi industriali. Ogni livello rappresenta il grado di protezione richiesto per un sistema o per una specifica parte dell'infrastruttura ed è definito sulla base di capacità e risorse a disposizione di un eventuale attaccante:

- **Security Level 0**: assenza di requisiti di sicurezza specifici;
- **Security Level 1**: protezione contro errori e utilizzi impropri del sistema;
- **Security Level 2**: protezione contro attacchi condotti con mezzi semplici, risorse limitate e competenze generiche;
- **Security Level 3**: protezione contro attacchi più sofisticati, condotti con risorse moderate e conoscenze specifiche dei sistemi di automazione;

- **Security Level 4**: protezione contro attacchi altamente sofisticati, condotti da attori dotati di risorse significative e competenze avanzate.

Questo approccio consente di adattare le misure di sicurezza in base al livello di minaccia che si intende mitigare.

2.2.2 NIST SP 800-82

Il NIST Special Publication 800-82 [16] è una pubblicazione del National Institute of Standards and Technology, intitolata *Guide to Operational Technology (OT) Security*, che rappresenta il principale riferimento per la sicurezza dei sistemi di controllo industriale. Questo documento fornisce linee guida sulle principali minacce informatiche e le contromisure raccomandate, evidenziando le principali differenze rispetto ai contesti IT tradizionali. I requisiti fondamentali su cui si concentra l'attenzione riguardano disponibilità, sicurezza operativa e continuità dei processi produttivi.

Una volta chiarito lo scopo principale del documento, si entra nei dettagli legati alla cybersecurity. In primo luogo viene rimarcata l'importanza del processo di gestione del rischio, evidenziando la necessità di una valutazione strutturata delle minacce e delle vulnerabilità prima di definire le misure di sicurezza da adottare. Questo processo si articola in diverse fasi tra cui l'identificazione delle risorse critiche, l'analisi delle minacce e delle vulnerabilità, la valutazione del rischio e la definizione delle contromisure. Viene esplicitata particolarmente la correlazione tra "safety" e "security", evidenziando ulteriormente le differenze con i sistemi IT e l'importanza della tutela dell'incolumità del personale e dei macchinari. Tale approccio consente di priorizzare gli interventi di sicurezza in funzione dell'impatto e della probabilità di occorrenza di determinati attacchi. Il documento sottolinea inoltre l'importanza di un approccio continuo alla gestione del rischio, in quanto le misure adottate devono essere periodicamente aggiornate in base all'evoluzione delle minacce e delle condizioni operative.

Il passaggio successivo fornisce dettagli per la progettazione e l'implementazione di architetture orientate alla sicurezza, introducendo il concetto di *defense-in-depth*, che prevede l'adozione di cinque livelli di protezione:

1. **Security Management**: definizione dei criteri di valutazione del rischio, delle priorità in termini di disponibilità, safety e continuità operativa, e delle modalità con cui tali requisiti vengono tradotti in politiche e controlli nei livelli successivi.

2. **Physical Security**: protezione fisica degli asset e dell'ambiente, elemento fondamentale nei sistemi OT, in quanto l'accesso diretto a PLC, armadi di rete o macchinari di produzione può introdurre vettori di attacco. Le misure includono protezione dei siti tramite barriere e controlli perimetrali, controllo degli accessi a locali e interfacce fisiche, e sistemi di monitoraggio quali telecamere e sensori. Viene inoltre sottolineata la necessità di coerenza tra politiche di accesso fisico e remoto, in quanto discrepanze tra i due casi possono comportare vulnerabilità.

3. **Network Security**: progettazione di architetture di rete segmentate, con l'obiettivo di isolare e proteggere i diversi componenti dei sistemi OT e IT. Un ruolo fondamentale è ricoperto dalla raccolta centralizzata dei log, che consente di monitorare gli eventi rilevanti per individuare comportamenti anomali. A tal fine vengono utilizzati strumenti quali *behavior anomaly detection*, *security information and event management (SIEM)*, *intrusion detection systems* e *intrusion prevention systems*, per generare allarmi in caso di operazioni riconducibili ad un attacchi informatici. In questo contesto viene inoltre considerato il paradigma Zero Trust come possibile evoluzione delle architetture di rete tradizionali.

4. **Hardware Security**: meccanismi di sicurezza integrati nei dispositivi, finalizzati a garantire l'integrità e l'identità dei sistemi, contribuendo alla costruzione di un modello di fiducia tra i componenti dell'infrastruttura. Tali meccanismi includono l'utilizzo di tecnologie hardware-based e algoritmi crittografici, come AES e SHA, compatibilmente ai requisiti di prestazione dei processi di produzione.

5. **Software Security**: misure di sicurezza a livello applicativo e di sistema, che comprendono:

    - **application allowlisting**, per prevenire l'esecuzione di software non autorizzato, indicando esplicitamente (tramite whitelist) le applicazioni consentite su ogni host;
    - **patching**, per mantenere i software costantemente aggiornati per mitigare vulnerabilità note, verificandone sempre la compatibilità nel sistema. Strumenti come web application

firewalls possono essere configurati per prevenire attacchi in caso di debolezze non corrette per il tempo necessario ad applicare gli aggiornamenti richiesti;

- **secure code developmemt**, per integrare la sicurezza nelle fasi di sviluppo del software;
- **configuration management**, per garantire configurazioni sicure dei sistemi, includendo l'uso di cifratura di dati archiviati e protocolli di comunicazione come TLS e IPSec.

In combinazione con il livello hardware, questo approccio consente di applicare il principio di *least functionality*, riducendo la superficie di attacco mediante la disabilitazione di servizi, porte, protocolli e software non necessari.

Il capitolo conclusivo del documento descrive l'applicazione del NIST Cybersecurity Framework (CSF) [15] ai sistemi OT, un insieme di linee guida progettato per supportare le organizzazioni nella gestione del rischio informatico in ambito IT. Il framework organizza le attività di sicurezza in cinque funzioni:

1. **Identificazione**: comprendere pienamente il sistema, identificando le risorse, le comunicazioni e le dipendenze tra i componenti dell'infrastruttura. In ambito OT questa fase risulta particolarmente critica, in quanto la conoscenza incompleta del funzionamento dei sistemi e delle loro interazioni può compromettere l'intero processo di gestione del rischio. Il primo passo consiste nell'identificazione degli asset, quali dispositivi, software, dati e personale coinvolti nelle operazioni industriali, e nel mantenimento di un inventario costantemente aggiornato. La comprensione dell'ambiente include inoltre la mappatura dei flussi di dati e la documentazione dell'architettura di rete. Tali informazioni costituiscono la base per lo sviluppo di una strategia di gestione del rischio, ovvero la definizione di criteri, priorità e livelli di tolleranza, nonché politiche, ruoli e responsabilità. Questo approccio guida le decisioni operative e l'adozione delle misure di sicurezza, che in ambito OT devono considerare anche le implicazioni sulla sicurezza fisica, sulla salute e sull'ambiente.

2. **Protezione**: implementare misure di sicurezza volte a limitare o prevenire accessi non autorizzati, quali il controllo degli accessi e la gestione delle identità. In ambito OT, tali misure devono essere progettate garantendo un equilibrio tra sicurezza e continuità operativa. Oltre ai dettagli sulla segmentazione della rete, già discussi in questo capitolo, viene evidenziata l'importanza dei meccanismi di autenticazione, per verificare l'identità degli utenti, incluso l'accesso a sistemi e dispositivi coinvolti nei processi produttivi. Vengono elencate diverse tecniche, tra cui l'utilizzo di token fisici, dati biometrici e smart card, spesso utilizzati in combinazione per aumentare il livello di sicurezza. L'autenticazione multi-fattore, in particolare, rappresenta una best practice, soprattutto per l'accesso remoto ai sistemi industriali. Tuttavia, l'adozione di tali soluzioni deve tenere conto dei limiti tecnologici di molti dispositivi OT, che spesso supportano esclusivamente meccanismi limitati. Viene specificato inoltre come l'autenticazione basata unicamente su password presenti diverse criticità, quali l'utilizzo di credenziali di default, la trasmissione in chiaro o la condivisione tra più utenti. Infine, questa funzione include attività di formazione e sensibilizzazione del personale, fondamentali per la prevenzione di incidenti, in particolare rispetto ad attacchi basati su ingegneria sociale. In ambito OT, tali attività devono essere adattate ai diversi ruoli operativi, includendo sia i sistemi informatici, sia i processi fisici controllati.

3. **Rilevazione**: implementare meccanismi per l'individuazione di anomalie o potenziali incidenti di sicurezza, attraverso attività di monitoraggio continuo dei sistemi e delle reti. In ambito OT, gli eventi soggetti ad analisi provengono da diverse fonti, quali sistemi, reti e accessi fisici, ponendo particolare attenzione a modifiche non autorizzate, comunicazioni inattese o la connessione di dispositivi non autorizzati. A tal fine, risultano necessari strumenti come sistemi centralizzati di gestione dei log, *SIEM*, *IDS* e *IPS*. In questo contesto è fondamentale definire il comportamento normale dell'infrastruttura, per distinguere attività lecite da potenziali minacce, cercando di ridurre al minimo il numero di falsi positivi. Tuttavia, in ambito OT è necessario valutare attentamente l'impatto di tali strumenti, in quanto attività come scansioni attive o agenti host-based possono interferire con il corretto funzionamento dei processi industriali.

4. **Risposta**: definire e attuare le azioni necessarie per contenere e gestire un incidente di

sicurezza una volta rilevato. Questa funzione si basa sulla predisposizione di piani di risposta strutturati, che descrivono le procedure da seguire, i ruoli e le responsabilità del personale coinvolto. Un elemento centrale è il coordinamento tra le diverse figure interne ed esterne all'organizzazione, includendo personale IT, OT e addetto alla sicurezza fisica, nonché, se necessario, fornitori o partner esterni potenzialmente coinvolti. La comunicazione deve essere pianificata in anticipo, prevedendo anche canali alternativi in caso di indisponibilità dei sistemi principali. Le attività di risposta comprendono inoltre l'analisi dell'incidente, finalizzata a comprenderne cause e impatti, supportando l'adozione di misure di mitigazione e il contenimento della propagazione. In ambito OT, tali azioni devono essere valutate con particolare attenzione, in quanto interventi come il blocco di sistemi o la disconnessione di reti possono influire sulla continuità operativa e sulla sicurezza fisica. Infine, è fondamentale prevedere un processo di miglioramento continuo, per aggiornare i piani di risposta e rafforzare la resilienza complessiva dell'infrastruttura.

5. **Recupero**: definire e attuare le attività necessarie per ripristinare le funzionalità dei sistemi e dei processi a seguito di un incidente di sicurezza, garantendo un ritorno tempestivo alle condizioni operative normali. Questa funzione include la predisposizione di piani di recupero, che definiscono procedure, priorità e tempi di ripristino, nonché il coordinamento tra le diverse figure coinvolte. Le attività comprendono il recupero dei dati, la verifica dell'integrità dei sistemi e la gestione delle comunicazioni interne ed esterne, necessarie per coordinare il processo e, ove richiesto, gestire gli aspetti legati alla reputazione. In ambito OT, anche in questo caso, è necessario considerare i vincoli di continuità operativa e sicurezza fisica ed è fondamentale aggiornare le strategie di recupero sulla base delle esperienze ottenute durante la gestione degli incidenti.

Il documento conclude presentando esempi di possibili fonti di minacce e di vulnerabilità nei sistemi OT, includendo infine casi storici di attacchi informatici reali su infrastrutture industriali. Tra questi assume particolare rilevanza Stuxnet [8], uno degli attacchi più noti ai sistemi di controllo industriale, che evidenzia le potenziali conseguenze fisiche ed economiche derivanti da un attacco mirato in questo contesto.

2.3 OpenZiti

OpenZiti [6] è una piattaforma open source progettata per implementare un modello di rete basato sul paradigma Zero Trust, rendendo i servizi accessibili esclusivamente a entità autenticate e autorizzate. In questo modello, le risorse non sono direttamente esposte sulla rete, risultando di fatto invisibili agli utenti non autorizzati. A differenza degli approcci tradizionali, ogni comunicazione viene sottoposta a processi di autenticazione, autorizzazione e cifratura. Un ulteriore vantaggio di questa soluzione è che non richiede infrastrutture hardware dedicate, essendo completamente implementabile via software, caratteristica che ne facilita l'adozione in contesti eterogenei.

2.3.1 Paradigma Zero Trust

Il paradigma Zero Trust rappresenta un'evoluzione dei modelli tradizionali di sicurezza informatica, nei quali la protezione dei sistemi è basata su un perimetro di rete considerato implicitamente fidato. Secondo quanto definito dal NIST [14], questo modello elimina tale assunzione, introducendo il principio secondo cui nessuna entità, interna o esterna alla rete, debba essere considerata attendibile a priori.

In questo contesto, ogni richiesta a una risorsa deve essere esplicitamente autenticata, autorizzata e continuamente verificata. Uno degli elementi centrali del modello Zero Trust è il concetto di *identity-based access*, in cui utenti, dispositivi e servizi sono associati ad identità univoche che costituiscono la base per la definizione delle politiche di accesso. Tali politiche seguono il principio del minimo privilegio, consentendo esclusivamente le comunicazioni strettamente necessarie allo svolgimento delle operazioni in base al ruolo delle entità coinvolte. L'accesso non è quindi più determinato dalla posizione nella rete, ma da un insieme di attributi associati all'identità del soggetto richiedente, allo stato del dispositivo e al contesto operativo.

Un ulteriore aspetto fondamentale è rappresentato dalla micro-segmentazione, che consente di suddividere l'infrastruttura in domini logici isolati, limitando la possibilità di movimento laterale da parte di un eventuale attaccante. Le comunicazioni tra i servizi non avvengono quindi liberamente all'interno della rete, ma sono consentite solo se esplicitamente autorizzate.

Il NIST identifica una serie di componenti architetturali chiave per l'implementazione di un sistema Zero Trust, tra cui il *Policy Decision Point* (PDP), responsabile della valutazione delle richieste

di accesso sulla base delle politiche definite, e il *Policy Enforcement Point* (PEP), incaricato di applicare le decisioni prese dal PDP. Nella pratica, PDP e PEP rappresentano un punto di controllo che verifica l'identità del soggetto e determina se esso sia autorizzato ad accedere alla risorsa richiesta. Le decisioni vengono prese sulla base di regole che possono considerare il contesto e lo stato del sistema, costituendo di fatto un punto di intermediazione tra un'*untrusted zone* e un'*implicit trust zone*. Il paradigma Zero Trust mira a ridurre al minimo queste zone di fiducia implicita. A tal fine, i meccanismi di controllo (PDP/PEP) vengono spostati il più vicino possibile alla specifica risorsa da proteggere, verificando la legittimità di ogni singola richiesta ed evitando che il superamento di un singolo controllo garantisca accesso indiscriminato ad altre risorse.

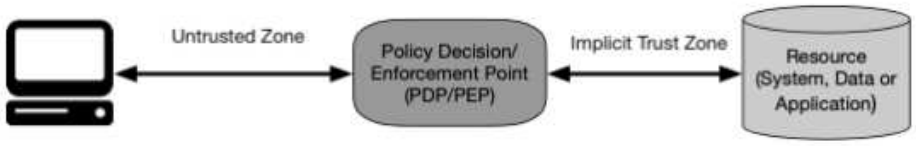

Figura 2.1: Accesso alle risorse nel modello Zero [14, Figura 1]

L'adozione del modello Zero Trust comporta quindi un cambiamento significativo rispetto agli approcci tradizionali, richiedendo l'integrazione di meccanismi di autenticazione robusti, sistemi di gestione delle identità e strumenti di monitoraggio avanzati. Tale approccio consente di ridurre significativamente la superficie di attacco e di migliorare la resilienza complessiva dell'infrastruttura. Soprattutto in ambito OT, caratterizzato da un'elevata eterogeneità, dalla presenza di dispositivi legacy e da una crescente interconnessione con reti IT, questo paradigma permette di superare i limiti delle architetture classiche, introducendo un livello di sicurezza idoneo a un contesto caratterizzato da un continuo incremento di minacce informatiche.

OpenZiti si inserisce in questo contesto come piattaforma che implementa nativamente tali principi, fornendo un sistema di comunicazione in cui i flussi sono mediati da meccanismi di autenticazione e autorizzazione basati sull'identità, e in cui le risorse non vengono mai esposte direttamente sulla rete.

2.3.2 Architettura

L'architettura di OpenZiti si basa sulla creazione di una rete overlay sicura, nella quale le comunicazioni tra i servizi non avvengono direttamente sull'infrastruttura IP sottostante, ma sono mediate da componenti dedicati che implementano i principi del paradigma Zero Trust. Questo approccio consente di disaccoppiare completamente l'accesso alle risorse dalla topologia della rete, introducendo un modello basato sull'identità e su politiche di accesso granulari. In ambito OT, tale modello consente di proteggere sistemi critici senza modificarne significativamente l'architettura di rete esistente, permettendo l'integrazione di dispositivi legacy e riducendo l'esposizione dei servizi industriali a potenziali attacchi.

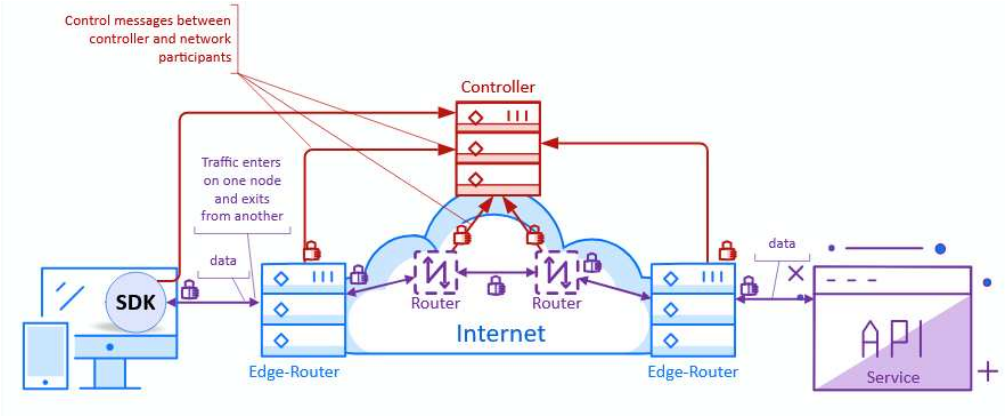

Figura 2.2: Rete overlay OpenZiti [6]

I principali componenti che costituiscono l'architettura di OpenZiti sono i seguenti:

- **Controller**: rappresenta il componente centrale dell'architettura OpenZiti, responsabile delle configurazioni dei servizi e dell'amministrazione delle identità associate a utenti, dispositivi e nodi della rete. Esso svolge inoltre il ruolo di *Policy Decision Point* (PDP), in quanto incaricato di autenticare e autorizzare ogni richiesta sulla base di politiche definite dall'amministratore del sistema. Garantisce quindi che ogni comunicazione avvenga esclusivamente tra entità legittimate, in accordo con i principi del paradigma Zero Trust. Per garantire la sicurezza delle comunicazioni, il Controller si basa su una *Public Key Infrastructure* (PKI), utilizzata per instaurare connessioni cifrate e mutuamente autenticate tramite protocollo TLS (mTLS). Una PKI è un insieme di tecnologie, procedure e componenti che consentono di gestire certificati

digitali e chiavi crittografiche, permettendo di associare un'identità a una chiave pubblica. Il ruolo centrale è svolto dalla *Certification Authority* (CA), ovvero l'entità responsabile dell'emissione e della firma dei certificati. La CA garantisce l'associazione tra un'identità e la relativa chiave pubblica, permettendo alle altre entità del sistema di verificare l'autenticità delle credenziali presentate. L'emissione di un certificato avviene tramite una *Certificate Signing Request* (CSR), attraverso la quale un'entità richiede alla CA la firma della propria chiave pubblica associata alla propria identità, per potersi poi autenticare all'interno della rete. OpenZiti permette di creare una CA interna gestita dal Controller, o di integrarsi con una PKI esterna già esistente. Le informazioni necessarie al funzionamento dell'overlay, quali configurazioni e identità, vengono salvate all'interno di un database locale, accentuando ulteriormente il ruolo di punto centrale di gestione dell'intera infrastruttura OpenZiti. Il Controller espone due principali interfacce di comunicazione, ciascuna dedicata a funzionalità specifiche. Nello specifico, una è un'interfaccia di gestione, accessibile dagli amministratori per diverse configurazioni, tra cui la definizione delle identità e delle politiche. La seconda è un'interfaccia operativa utilizzata dai nodi della rete overlay per le funzioni di autenticazione e autorizzazione.

- **Router**: rappresenta il componente incaricato del trasferimento dei dati all'interno della rete overlay ed è responsabile dell'instradamento del traffico tra le diverse entità della rete, garantendo comunicazioni sicure e affidabili tra i nodi. I Router sono interconnessi tra loro formando una *mesh network*, nella quale vengono costantemente monitorati parametri come la latenza e la disponibilità dei collegamenti. Questo consente di selezionare dinamicamente i cammini più efficienti per il trasporto dei dati, garantendo l'operatività del sistema anche in caso di guasti o indisponibilità di uno o più nodi. Il Router rappresenta inoltre il punto di ingresso alla rete overlay per i client, permettendo loro l'accesso ai servizi. In coordinamento con il Controller, contribuisce ai processi di autenticazione e autorizzazione, applicando le politiche di accesso. In questo contesto, il Router può essere interpretato come un *Policy Enforcement Point* (PEP), in quanto incaricato di applicare le decisioni definite sul Controller, garantendo lo scambio di informazioni esclusivamente tra entità autorizzate. Il Router espone inoltre due interfacce di comunicazione. Una è dedicata alla connessione con il Controller, per la gestione delle operazioni di autorizzazione e di autenticazione, e per il controllo delle

politiche da applicare. L'altra è utilizzata per la comunicazione con gli altri nodi della rete. Ogni Router deve essere associato ad una propria identità univoca, sono quindi anch'essi soggetti a limiti di accesso in base alle politiche definite dagli amministratori. Questo approccio rafforza ulteriormente il modello di sicurezza basato sull'identità, estendendolo anche ai componenti infrastrutturali della rete.

- **Edge Clients**: rappresentano il componente attraverso cui utenti e applicazioni accedono alla rete overlay OpenZiti. Gli Edge Clients consentono di stabilire connessioni sicure verso uno o più servizi, sfruttando i meccanismi di autenticazione e autorizzazione gestiti dal Controller e instradati tramite i Router. Sono supportate due principali modalità di integrazione. Nel caso di nuove applicazioni (*greenfield*), è possibile utilizzare gli *SDK* messi a disposizione dalla piattaforma, che permettono lo sviluppo di applicazioni nativamente compatibili con OpenZiti. Per applicazioni esistenti (*brownfield*), vengono forniti invece componenti denominati *tunneler*, che consentono l'accesso alla rete overlay senza richiedere significative modifiche al codice. I tunneler operano a livello di sistema, intercettando il traffico e instradandolo verso i Router Ziti, garantendo così una transizione trasparente verso un modello Zero Trust. Essi sono associati ad un'identità e sono configurabili anche come *host* per esporre servizi *brownfield* sulla rete sicura OpenZiti. Quest'ultimo approccio risulta particolarmente adatto in contesti OT caratterizzati dall'adozione dei paradigmi dell'industria 4.0.

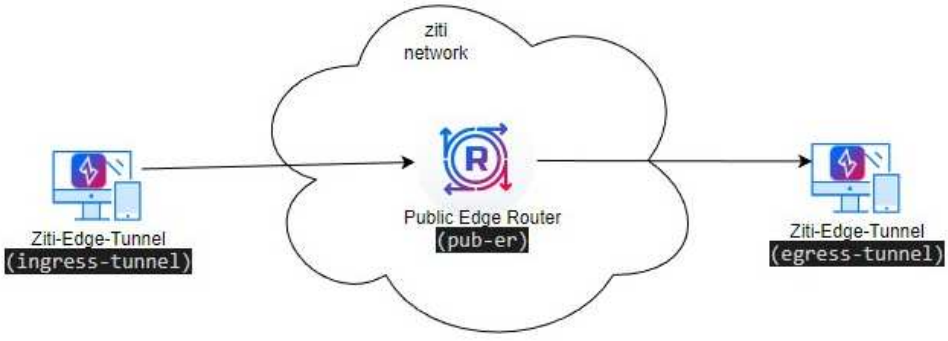

Figura 2.3: Diagramma di rete OpenZiti con tunneler [6]

Una volta definiti i componenti dell'architettura, è necessario considerare anche gli elementi logici che ne regolano il funzionamento:

- **Servizi**: rappresentano le risorse accessibili all'interno della rete overlay OpenZiti, introducendo un livello di astrazione rispetto ai tradizionali concetti di rete basati su indirizzi IP o nomi DNS. Un servizio è associato a un'identità e a un nome logico, accessibile ai client autorizzati indipendentemente dalla posizione o dalla configurazione dell'infrastruttura sottostante. Un aspetto fondamentale è il concetto di *service termination*, ovvero il modo in cui il traffico raggiunge l'applicazione che eroga l'effettivo servizio. Sono supportate diverse modalità di terminazione, che variano in base al livello di integrazione con la piattaforma. Nel caso di applicazioni sviluppate includendo direttamente gli SDK, è possibile ottenere un modello *end-to-end Zero Trust*, ovvero comunicazioni completamente cifrate. In alternativa, per applicazioni esistenti, è necessario utilizzare un proxy o un Router come terminatori, accettando diversi compromessi. Poiché la cifratura del traffico è garantita unicamente fino al punto di terminazione, l'utilizzo di un Router esterno al nodo che ospita il servizio comporta l'esposizione di informazioni non cifrate sulla rete. Nei sistemi *brownfield*, una soluzione preferibile consiste nell'utilizzo di un proxy, ovvero un tunneler installato direttamente sul nodo, che inoltra le comunicazioni non cifrate verso un'interfaccia locale, evitando la trasmissione di dati sensibili sulla rete. In questo modello i servizi risultano completamente invisibili, in quanto non espongono direttamente porte, ma instaurano esclusivamente connessioni verso l'infrastruttura OpenZiti. L'accesso ad essi è quindi regolato esclusivamente dalle politiche di sicurezza e dalle identità coinvolte, rafforzando i principi del paradigma Zero Trust, riducendo significativamente la superficie d'attacco.

- **Identità**: rappresentano le entità attraverso le quali utenti, dispositivi e applicazioni possono autenticarsi all'interno della rete OpenZiti. Ogni identità è associata ad una coppia di chiavi crittografiche e a un certificato X.509. Ogni connessione nella rete overlay utilizza il protocollo *mutual TLS* (mTLS), che consente di verificare l'autenticità di entrambe le parti coinvolte nella comunicazione. Client e servizio presentano reciprocamente il proprio certificato firmato dalla CA prima di instaurare la connessione. Le identità sono gestite dal Controller e possono essere considerate analoghe agli account utente nei sistemi tradizionali,

estese ad ogni entità fisica e logica della rete OpenZiti. Il processo mediante il quale un'entità ottiene il proprio certificato e viene associata ad un'identità della rete è denominato *enrollment*. La modalità principale in cui questo avviene è tramite l'utilizzo di un *One Time Token* (OTT), contenuto in un *JSON Web Token* JWT, generato alla creazione dell'identità. Un JWT è un file strutturato in tre parti: un header, che specifica l'algoritmo di firma digitale, un payload contenente le informazioni sull'identità (tra cui il OTT), e la signature del Controller per assicurare l'integrità del file. Questo meccanismo garantisce che solo entità legittime possano ottenere credenziali valide per l'accesso alla rete. Le identità costituiscono inoltre la base per i meccanismi di autorizzazione, definiti dalle politiche di accesso.

- **Politiche**: definiscono le regole che governano le interazioni tra identità, servizi e Router all'interno della rete OpenZiti. Rappresentano il meccanismo attraverso cui viene implementato il controllo degli accessi, stabilendo quali entità possono comunicare tra loro e in quali modalità. Le politiche possono definire esplicitamente le identità coinvolte in una regola o utilizzare i *role attributes*, ovvero attributi associati alle entità che consentono di raggrupparle logicamente. Questo approccio consente una gestione scalabile e flessibile delle configurazioni, evitando la definizione di politiche per ogni singola entità della rete. Esistono diverse tipologie di politiche, tra cui:
  - **Service Policies**, che definiscono le relazioni tra identità e servizi, determinando quali entità possono utilizzare o erogare una determinata risorsa;
  - **Edge Router Policies**, che stabiliscono quali identità possono accedere alla rete overlay tramite specifici Router;
  - **Service Edge Router Policies**, che regolano quali servizi possono essere esposti o raggiunti tramite determinati Router.

  Attraverso le politiche, OpenZiti realizza un modello di sicurezza granulare basato sull'identità, in cui ogni accesso è esplicitamente autorizzato e verificato, in linea con i principi del paradigma Zero Trust.

2.4 WebAuthn

WebAuthn [1] è uno standard aperto sviluppato dalla *FIDO Alliance* e dal *World Wide Web Consortium* (W3C), progettato per fornire un meccanismo di autenticazione forte e sicuro per applicazioni web. Tale standard è parte integrante del framework FIDO2 [2], un insieme di specifiche costituito da due componenti principali: WebAuthn, che definisce l'interfaccia tra applicazioni web e browser, e il protocollo *Client to Authenticator Protocol* (CTAP), che regola la comunicazione tra il client e l'authenticator. Questa architettura consente di autenticare gli utenti senza password tradizionali, riducendo significativamente i rischi associati all'utilizzo di credenziali deboli, riutilizzate o compromesse. I sistemi di autenticazione basati su password presentano numerose criticità, tra cui l'esposizione ad attacchi di phishing, brute-force e credential stuffing. WebAuthn introduce un modello di autenticazione basato sulla crittografia asimmetrica, in cui non vi è alcuna condivisione di segreti con il server, ma le credenziali sono gestite e protette all'interno di dispositivi sicuri controllati dall'utente.

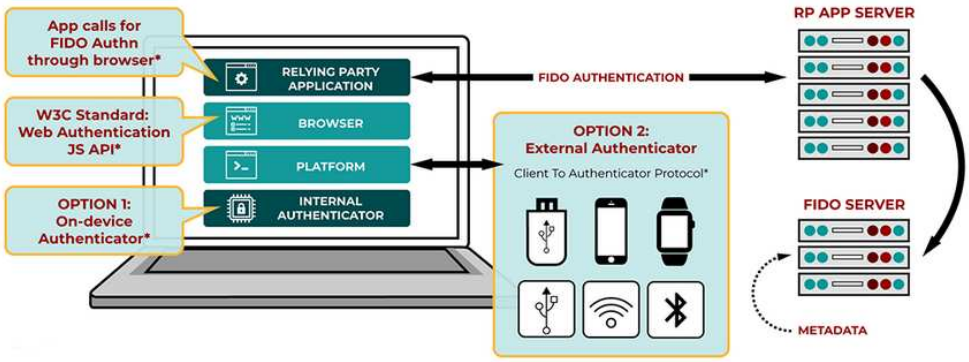

Figura 2.4: Progetto FIDO2 [2]

I tre attori principali coinvolti nel funzionamento del protocollo sono:

- **Client**: rappresenta l'applicazione utilizzata dall'utente, tipicamente un browser web, che gestisce l'interazione con il server e con l'authenticator;
- **Relying Party**: rappresenta il server dell'applicazione che richiede l'autenticazione dell'utente e verifica la validità delle credenziali presentate;

- **Authenticator**: rappresenta il dispositivo che genera e protegge le chiavi crittografiche, come una chiave di sicurezza hardware, un modulo TPM o un sensore biometrico integrato nel dispositivo dell'utente.

WebAuthn definisce quindi i processi di registrazione e autenticazione:

- **Registrazione (sign-up)**: rappresenta il processo mediante il quale un client associa un authenticator ad un account presso una Relying Party. Durante questa fase viene creata una coppia di chiavi crittografiche, che costituirà la base per le successive operazioni. Il processo ha inizio quando l'applicazione richiede una nuova credenziale, generando una *challenge*, ovvero una sequenza di bit casuale e univoca. Tale valore viene inviato al client, che lo inoltra all'authenticator insieme a informazioni relative al dominio dell'applicazione e all'identità dell'utente. A questo punto, l'authenticator genera la coppia di chiavi asimmetriche: una privata, memorizzata in modo sicuro all'interno del dispositivo che ospita l'authenticator, e una pubblica. La *challenge* viene quindi firmata mediante la chiave privata e restituita al client insieme alla chiave pubblica, tramite la quale il server può verificare la validità della risposta e, in caso positivo, memorizzare la chiave pubblica associata all'utente. In questo modo la chiave privata non lascia mai l'authenticator e viene utilizzata esclusivamente per generare firme digitali, garantendo la protezione delle credenziali. Al termine di questa fase, il dispositivo di autenticazione risulta registrato e potrà essere utilizzato per le successive richieste di accesso. Si stabilisce quindi un legame forte tra l'identità dell'utente, il servizio e l'authenticator, senza la necessità di utilizzare password.

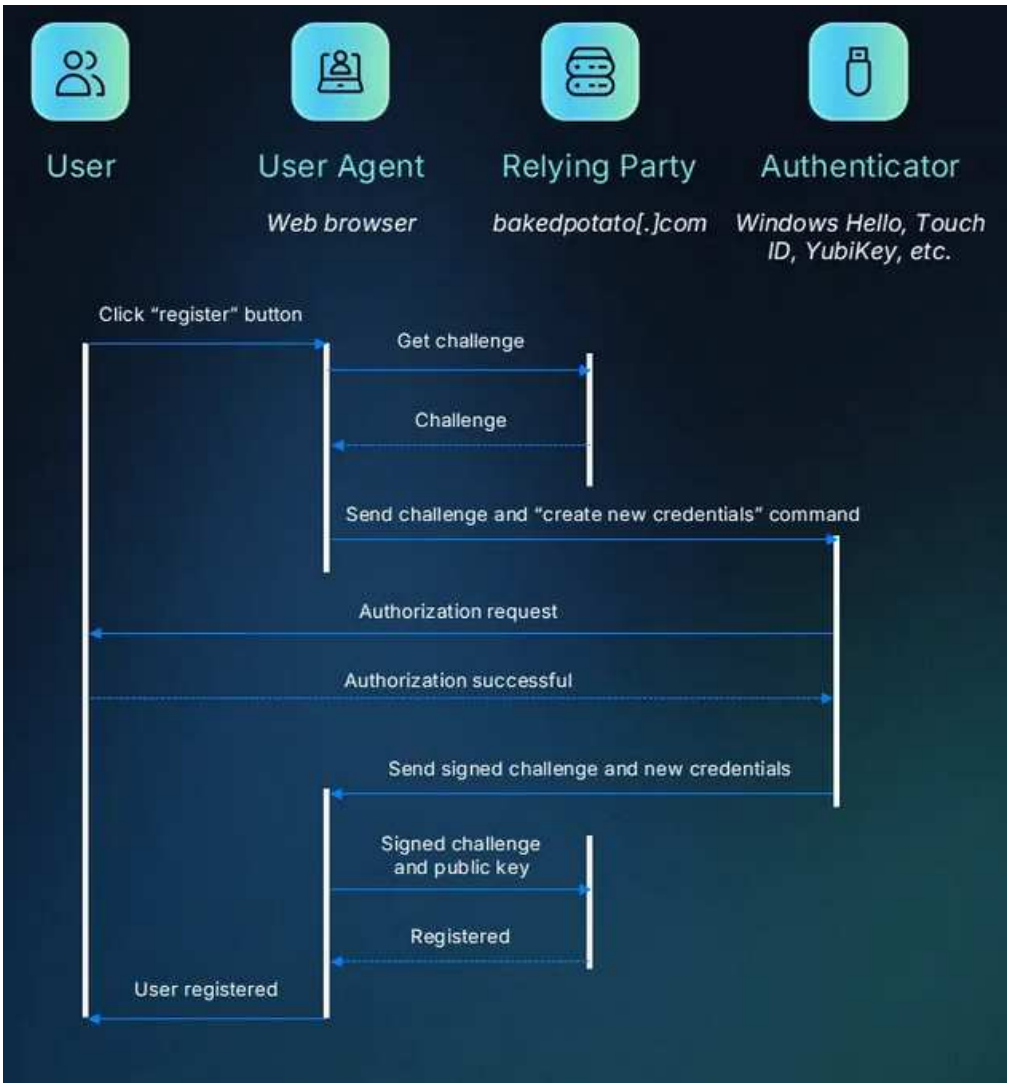

Figura 2.5: Flusso di registrazione WebAuthn [5]

- **Autenticazione (login)**: rappresenta il processo mediante il quale un utente dimostra la propria identità utilizzando un authenticator precedentemente registrato. Il processo ha inizio quando l'applicazione richiede l'autenticazione, generando una *challenge*, valida per un intervallo di tempo limitato (*timeout*). Il Relying Party può inoltre specificare un insieme di credenziali ammesse tramite il parametro *allowCredentials*, ovvero una lista di identificativi delle chiavi associate all'utente registrato. A questo punto, l'authenticator utilizza la chiave privata generata durante la fase di registrazione per firmare la challenge. Il Relying Party verifica la risposta utilizzando la chiave pubblica associata all'utente, precedentemente salvata. Se la verifica ha esito positivo, l'utente viene autenticato e può accedere al servizio protetto.

Questo meccanismo consente di dimostrare il possesso della chiave privata senza che essa venga mai esposta, garantendo un elevato livello di sicurezza. Poiché la risposta è legata alla challenge univoca e al dominio del servizio, vengono prevenuti attacchi di tipo replay e phishing.

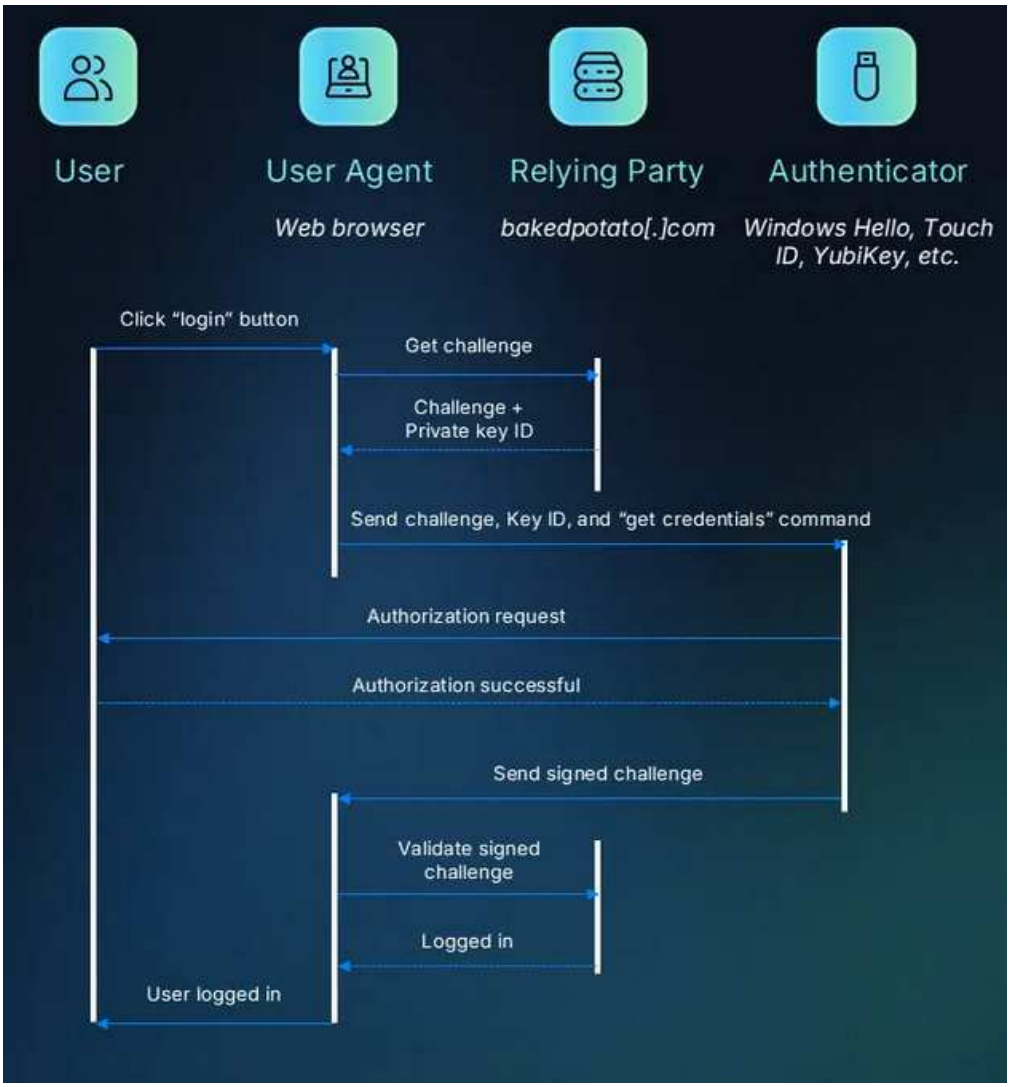

Figura 2.6: Flusso di autenticazione WebAuthn [5]

WebAuthn rappresenta quindi un'evoluzione significativa rispetto ai tradizionali sistemi di autenticazione, introducendo un modello più sicuro e affidabile. Questo approccio consente di rafforzare i meccanismi di controllo degli accessi, eliminando la dipendenza dalle password e migliorando al contempo la sicurezza e l'esperienza dell'utente. Il protocollo è inoltre progettato per essere com-

patibile con diversi dispositivi e browser, favorendone l'adozione in molteplici contesti applicativi e facilitando l'integrazione sia per gli utenti finali sia per gli sviluppatori.

2.5 Problemi

Nel contesto attuale, caratterizzato da una crescente interconnessione tra sistemi informatici e infrastrutture industriali, la sicurezza rappresenta una delle principali criticità da affrontare. In particolare, gli ambienti OT presentano una serie di sfide specifiche dovute alla presenza di sistemi legacy, requisiti elevati di disponibilità e vincoli operativi che limitano l'adozione di soluzioni di sicurezza tradizionali. I modelli di sicurezza convenzionali, basati sul concetto di perimetro fidato, risultano spesso inadeguati nel fronteggiare le minacce moderne. L'assunzione che le entità interne alla rete siano affidabili espone i sistemi a rischi significativi, specialmente in presenza di accessi remoti, integrazione con sistemi IT e crescente digitalizzazione dei processi industriali. Parallelamente, i meccanismi di autenticazione tradizionali, basati principalmente su credenziali statiche come username e password, presentano evidenti problematiche. Tali difficoltà risultano particolarmente critiche in ambito OT, dove la sicurezza deve essere bilanciata con requisiti di continuità operativa e dove le pratiche di cybersecurity non sono sempre applicate in modo rigoroso. Alla luce di queste considerazioni, emerge la necessità di adottare nuovi approcci alla sicurezza, in grado di superare i limiti dei modelli tradizionali e di adattarsi alle specificità degli ambienti industriali. Nel presente capitolo verranno analizzate nel dettaglio le principali problematiche legate ai sistemi esistenti, al fine di stabilire i requisiti necessari per la progettazione di una soluzione sicura ed efficace.

2.5.1 Autenticazione

I meccanismi di autenticazione tradizionali, basati principalmente sull'utilizzo di credenziali statiche come password o PIN, rappresentano uno dei principali punti deboli nei sistemi informativi. Tali credenziali sono spesso soggette a pratiche insicure, come la scelta di password deboli, il riutilizzo tra più servizi e la condivisione tra utenti, aumentando significativamente il rischio di compromissione. Di conseguenza, questi sistemi risultano particolarmente vulnerabili a diverse tipologie di attacchi,

tra cui:

- **phishing**: rappresenta una delle tecniche più diffuse ed efficaci. Si tratta di un attacco di ingegneria sociale, ovvero una categoria di minacce che si basa sulla manipolazione del comportamento umano piuttosto che sull'individuazione di vulnerabilità tecniche nei sistemi informatici. Lo scopo è quello di indurre un utente a fornire volontariamente le proprie credenziali, simulando un contesto legittimo. Il funzionamento tipico di un attacco di phishing prevede la creazione di una pagina web contraffatta che replica l'interfaccia di un servizio autentico, come un portale aziendale o un sistema di accesso remoto. L'utente viene quindi indotto a visitare tale pagina tramite comunicazioni ingannevoli, come email o messaggi contenenti link fraudolenti. Una volta inserite le credenziali, queste vengono intercettate dall'attaccante, che può successivamente utilizzarle per accedere ai sistemi legittimi.

- **brute-force**: rappresenta una tecnica di compromissione delle credenziali basata sul tentativo sistematico e automatico di tutte le possibili combinazioni di caratteri fino all'individuazione di quella corretta. Questo tipo di attacco sfrutta la limitata complessità delle password scelte dagli utenti e la crescente disponibilità di risorse computazionali. Una variante particolarmente efficace è il brute-forcing basato su dizionari (*dictionary attack*), in cui l'attaccante utilizza liste predefinite di combinazioni frequentemente utilizzate, riducendo significativamente il numero di tentativi necessari. In molti casi, infatti, gli utenti tendono a scegliere credenziali prevedibili o basate su schemi semplici. In ambito OT, la presenza di sistemi legacy, configurazioni non aggiornate e la tendenza ad utilizzare password corte e facili da ricordare per semplificare le attività operative possono facilitare ulteriormente questo tipo di attacco.

- **credential stuffing**: rappresenta una tecnica di compromissione delle credenziali basata sull'utilizzo di coppie username/password precedentemente esposte in seguito a violazioni di dati anche su altri sistemi (*data breach*). Questo tipo di attacco risulta particolarmente efficace a causa della diffusa pratica di riutilizzo delle password da parte degli utenti su più piattaforme. Un attaccante può quindi automatizzare l'invio di grandi quantità di richieste di autenticazione verso diversi servizi, verificando rapidamente su quali di questi le credenziali risultano valide. Poiché gli username e password utilizzati sono corretti, tali attacchi risultano

molto difficili da rilevare rispetto a tecniche basate su tentativi casuali. In ambito OT, il rischio è ulteriormente amplificato da pratiche non sempre rigorose, come la condivisione di account. L'utilizzo di credenziali compromesse può quindi consentire l'accesso a sistemi critici senza destare sospetti.

Spesso quindi, la sicurezza dell'intero sistema dipende esclusivamente dalla robustezza delle password scelte dagli utenti, rendendo tale modello intrinsecamente fragile. Un ulteriore limite è rappresentato dalla difficoltà nella gestione delle credenziali, che possono essere dimenticate, con conseguente rallentamento nelle normali attività di produzione, o salvate su supporti non protetti. Alla luce di queste considerazioni, emerge la necessità di adottare meccanismi di autenticazione più robusti, basati su modelli che eliminino la dipendenza dalle credenziali statiche e che garantiscano un livello di sicurezza adeguato e resiliente, anche in presenza di minacce avanzate.

2.5.2 Modelli di rete

I modelli di sicurezza tradizionali si basano sul concetto di perimetro di rete, secondo il quale l'infrastruttura viene suddivisa in una zona interna, considerata affidabile, e una esterna, ritenuta non fidata. In questo approccio, i sistemi di difesa come firewall e sistemi di rilevamento delle intrusioni, sono posizionati ai confini della rete con l'obiettivo di filtrare il traffico in ingresso e impedire accessi non autorizzati. Questo modello, sebbene efficace in infrastrutture tradizionali, presenta numerosi limiti, soprattutto in un contesto di integrazione tra ambienti OT e IT. L'assunzione implicita di fiducia nei confronti delle entità interne alla rete rappresenta uno dei principali punti deboli. Una volta ottenuto un accesso iniziale a un sistema, un attaccante può tentare di espandere il proprio controllo compromettendo ulteriori nodi. Questo processo è noto come movimento laterale (*lateral movement*) e tale modello consente all'attaccante di raggiungere con relativa facilità sistemi critici all'interno della rete, a seguito della compromissione di un singolo servizio. Inoltre, la diffusione di dispositivi eterogenei, sistemi legacy e architetture distribuite rende sempre più complesso definire un perimetro chiaro e ben delimitato. In ambito OT, tali problematiche risultano ulteriormente accentuate, poiché molte infrastrutture industriali non sono progettate con requisiti di sicurezza moderni e molti dispositivi non dispongono delle risorse necessarie per implementare soluzioni avanzate. Un ulteriore limite è rappresentato dalla scarsa granularità nel controllo degli accessi:

una volta autenticato, un utente è spesso in grado di accedere a un'ampia porzione della rete, aumentando il rischio di compromissione dei sistemi in caso di furto di credenziali. Con la crescente digitalizzazione e integrazione con sistemi IT, il numero di servizi esposti sulla rete tende ad aumentare, determinando un'espansione della superficie d'attacco. Questo rende il processo di hardening, ovvero un insieme di azioni volte a rendere sicuro ogni accesso al sistema, lungo, complesso, costoso e soggetto a errori. Un altro aspetto critico nei modelli tradizionali riguarda l'assenza di meccanismi di segmentazione fine-graned, che consentono di isolare le risorse e limitare le comunicazioni tra i diversi componenti.

Risulta quindi evidente la necessità di adottare modelli di sicurezza alternativi, in grado di superare il concetto di fiducia implicita e di garantire un controllo più rigoroso e dinamico degli accessi alle risorse.

3. Analisi di una Linea Pilota

Al fine di valutare concretamente le criticità descritte nei capitoli precedenti, è stata condotta un'analisi su una linea pilota reale in ambito industriale. L'obiettivo è osservare come le problematiche relative all'autenticazione e ai modelli tradizionali di rete si manifestino in un contesto operativo caratterizzato da vincoli tipici degli ambienti OT.
L'analisi si concentra sull'architettura della linea, sulle modalità di accesso ai sistemi e sui meccanismi di sicurezza adottati, con particolare attenzione agli eventuali punti di debolezza.
I risultati ottenuti costituiscono la base per la definizione dei requisiti di sicurezza della soluzione proposta, che verrà sviluppata e valutata nei capitoli successivi.

3.1 Descrizione della Linea

La linea pilota analizzata è costituita da un sistema automatizzato per l'assemblaggio e il riempimento di fialette contenenti una soluzione medica. L'infrastruttura è dotata di un controllore logico programmabile (PLC) Siemens, responsabile della gestione e del coordinamento delle operazioni lungo la linea.
La movimentazione dei materiali è realizzata tramite quattro azionamenti industriali, collegati ai rispettivi motori e utilizzati per il controllo dei nastri trasportatori.
La linea include inoltre quattro robot industriali e cobot collaborativi, impiegati per operazioni di manipolazione e assemblaggio. I relativi Controller ABB sono dotati di interfacce di rete e, in alcuni casi, espongono interfacce web per la configurazione e la manutenzione.
Dal punto di vista della supervisione, è presente un sistema SCADA/HMI, utilizzato per il monitoraggio dello stato della linea e per l'interazione con il PLC. Il sistema consente la visualizzazione delle variabili di processo e la modifica dei parametri operativi.
L'infrastruttura è completata da dispositivi di supporto, tra cui moduli di I/O distribuiti e punti di accesso alla rete per attività di manutenzione, che permettono la connessione diretta ai dispositivi di controllo.

Nel complesso, la linea presenta un'architettura distribuita, in cui dispositivi eterogenei comunicano attraverso la rete ed espongono servizi accessibili sulla rete locale, spesso in assenza di adeguati meccanismi di isolamento e controllo degli accessi.

3.2 Problemi

Al fine di valutare il livello di esposizione della linea pilota, è stata condotta un'attività di scansione della rete e intercettazione del traffico, assumendo il punto di vista di un potenziale attaccante con accesso alla `LAN`.

3.2.1 Scansione della rete

È stata effettuata una scansione completa mediante lo strumento `nmap`, utilizzato per identificare gli host attivi e i servizi esposti sulle rispettive porte TCP. La scansione è stata eseguita con opzioni avanzate, includendo il rilevamento dei servizi e delle versioni, al fine di ottenere una visione dettagliata della superficie di attacco.

```
nmap -A -p 0-65535 192.168.14.0/24
```

La scansione evidenzia come una parte significativa dei dispositivi della linea risulti direttamente individuabile sulla rete. Ciò costituisce già di per sé una debolezza, in quanto consente a un potenziale attaccante di ricostruire rapidamente l'infrastruttura e di identificare i principali punti di accesso ai servizi esposti.

Particolarmente rilevante è la presenza di dispositivi industriali direttamente raggiungibili in rete, come il nodo `192.168.14.200`, identificato come PLC Siemens con servizio S7 sulla porta `102/tcp` e servizio OPC UA sulla porta `4840/tcp`.

```
Nmap scan report for 192.168.14.200
Host is up (0.00051s latency).
Not shown: 65534 closed tcp ports (conn-refused)
PORT     STATE SERVICE    VERSION
102/tcp  open  iso-tsap   Siemens S7 PLC
| s7-info:
```

```
|   Module: 6ES7 510-1SJ01-0AB0
|   Basic Hardware: 6ES7 510-1SJ01-0AB0
|   Version: 2.9.7
|   System Name: ET 200SP station_1
|   Module Type: A10021
|   Serial Number: S C-RNA0YH5F2023
|   Plant Identification:
|_  Copyright: Original Siemens Equipment
4840/tcp open  opcua-tcp?
Service Info: Device: specialized
```

La presenza del servizio S7 potrebbe rappresentare una criticità rilevante, in quanto rende il PLC direttamente raggiungibile dalla rete locale. Tale porta è infatti associata al trasporto ISO-TSAP utilizzato dalle comunicazioni industriali Siemens e, in assenza di adeguate misure di protezione, può consentire a un host presente sulla stessa rete di tentare l'interrogazione del controllore. La sicurezza dell'accesso non è quindi garantita dal protocollo stesso, ma dipende dalle configurazioni di protezione del PLC e dai meccanismi di segmentazione e filtraggio adottati a livello di rete.
Diversamente dal servizio S7, la presenza di un endpoint OPC UA è meno probabile implichi la possibilità di accesso ai dati da parte di un attaccante, poiché il protocollo prevede meccanismi di sicurezza configurabili, quali security policy, cifratura dei messaggi e autenticazione degli utenti. Tuttavia, l'esposizione della porta rende comunque il servizio raggiungibile e identificabile dalla rete locale, consentendo a un potenziale attaccante di enumerare l'endpoint e di tentare l'accesso qualora siano state adottate configurazioni deboli o modalità prive di protezione.

Un'ulteriore informazione interessante riguarda gli host `192.168.14.6`, `192.168.14.7` e `192.168.14.8`, che corrispondono agli indirizzi IP assegnati ai bracci robotici. Essi espongono tutti il servizio SSH sulla porta `22/tcp` e condividono le stesse chiavi host SSH, suggerendo che siano stati clonati a partire dalla medesima immagine senza rigenerazione delle credenziali crittografiche. Pur non costituendo di per sé una vulnerabilità direttamente sfruttabile, questa configurazione rappresenta comunque una debolezza. In caso di compromissione di uno dei nodi, la condivisione della stessa chiave host SSH ridurrebbe l'affidabilità dell'identificazione dei dispositivi e potrebbe facilitare attività di impersonificazione o movimenti laterali all'interno dell'infrastruttura.

```
Post-scan script results:
| ssh-hostkey: Possible duplicate hosts
| Key 256 e1:cd:40:72:50:30:f0:2c:e8:5e:96:5b:57:bb:aa:0e (ED25519) used by:
|   192.168.14.6
|   192.168.14.7
|   192.168.14.8
| Key 256 94:e7:2e:1a:d6:e5:b6:ad:7d:9c:3a:64:44:6f:ac:dc (ECDSA) used by:
|   192.168.14.6
|   192.168.14.7
|   192.168.14.8
| Key 2048 d1:43:88:14:13:1b:6c:40:81:e3:62:bc:bb:9f:12:c1 (RSA) used by:
|   192.168.14.6
|   192.168.14.7
|_  192.168.14.8
```

Le problematiche discusse rappresentano soltanto una parte della superficie di attacco osservata. La scansione evidenzia infatti la presenza di numerosi altri servizi esposti, che, pur non essendo stati analizzati singolarmente, ampliano ulteriormente le possibilità di ricognizione e di attacco.
Ne consegue l'importanza di adottare modelli in cui i servizi non siano direttamente visibili né raggiungibili dalla rete locale, ma risultino accessibili esclusivamente attraverso meccanismi espliciti di autenticazione, autorizzazione e mediazione del traffico.

3.2.2 Intercettazione del traffico

Un'ulteriore criticità emersa dall'analisi riguarda la possibilità di ottenere accesso alla rete assumendo l'identità di un dispositivo già presente nell'infrastruttura. In particolare, il servizio DHCP interno assegna gli indirizzi IP in funzione del MAC address del nodo richiedente. Di conseguenza, modificando opportunamente il proprio MAC address, un attaccante può ottenere l'assegnazione dell'indirizzo IP associato a un dispositivo legittimo e inserirsi nel dominio di comunicazione della linea. Questa possibilità evidenzia una debolezza strutturale dei modelli tradizionali basati sulla

sola appartenenza alla LAN: l'identità del nodo è implicitamente dedotta dalla sua presenza fisica sulla rete, senza ulteriori garanzie circa la sua autenticità.
Tale condizione è stata sfruttata sperimentalmente per assumere l'identità di diversi componenti della linea e osservare le comunicazioni scambiate tra i nodi dell'infrastruttura. In questo modo è stato possibile intercettare traffico applicativo e di controllo, confermando come un attaccante con accesso alla rete locale possa non solo enumerare i dispositivi presenti, ma anche inserirsi attivamente nei flussi di comunicazione. L'attività di intercettazione ha inoltre evidenziato come una parte significativa delle comunicazioni presenti sulla rete non fosse cifrata. Ciò ha consentito di osservare direttamente porzioni di traffico e di ricavare informazioni utili sul funzionamento dell'infrastruttura, confermando che molti protocolli e servizi in uso si basano ancora su un modello di fiducia implicita nella rete locale, piuttosto che sulla protezione crittografica dei dati scambiati.

Nella Figura 3.1 è possibile osservare uno scambio di pacchetti CIP tra l'host `192.168.14.5`, associato al controller ABB, e l'host `192.168.14.8`, riconducibile a un dispositivo robotico della linea. Tale evidenza suggerisce la presenza di comunicazioni industriali effettive tra il controller e il dispositivo di campo e conferma che la rete locale trasporta non solo traffico di supervisione e configurazione, ma anche flussi direttamente coinvolti nel funzionamento operativo del sistema.

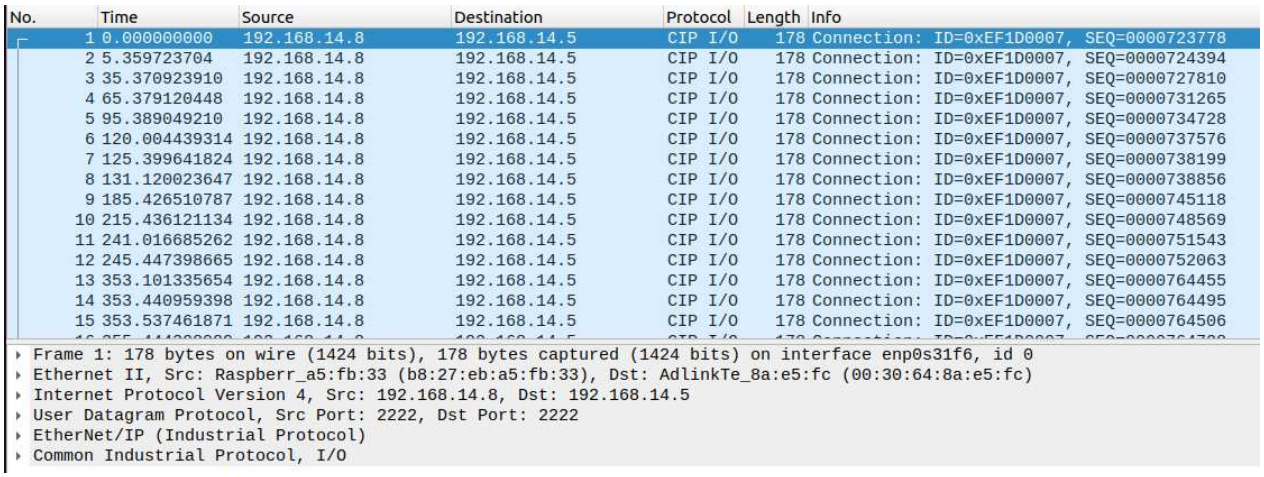

Figura 3.1: Intercettazione comunicazioni tra Controller ABB e dispositivo robotico

Un ulteriore esempio è mostrato nella Figura 3.2, in cui è riportata la cattura di traffico PROFINET tra il PLC Siemens e un controller ABB. In questo caso, Wireshark identifica esplicitamente i

pacchetti come PNIO-CM, evidenziando richieste di connessione tra i due nodi. Anche questa evidenza conferma la presenza di traffico industriale reale tra componenti critici della linea e mostra come tali comunicazioni possano essere osservate da un attaccante presente sulla rete locale.

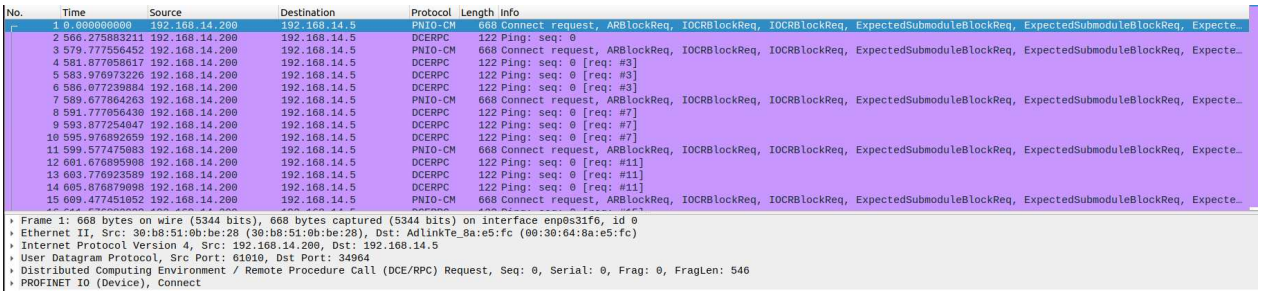

Figura 3.2: Intercettazione comunicazioni tra PLC e Controller ABB

3.2.3 Credenziali non sicure

Un'ulteriore criticità riguarda i meccanismi di autenticazione adottati per l'accesso alle interfacce di gestione dei dispositivi. In particolare, le interfacce web esposte dai controller ABB risultano accessibili mediante credenziali di default. Analogamente, la programmazione dei robot avviene tramite un tool proprietario, al quale è stato possibile accedere utilizzando credenziali predefinite. Questo aspetto evidenzia come, in assenza di una corretta gestione delle credenziali, anche la presenza di interfacce protette da autenticazione non costituisca una misura di sicurezza sufficiente. Un attaccante con accesso alla rete locale potrebbe infatti non solo individuare tali servizi, ma anche autenticarsi con privilegi legittimi, ottenendo la possibilità di consultare configurazioni, modificare parametri operativi o intervenire sul comportamento dei dispositivi.

4. Progettazione

Alla luce delle criticità analizzate nei capitoli precedenti, emerge la necessità di definire un'architettura di sicurezza in grado di superare i limiti dei modelli tradizionali. Il presente capitolo descrive gli obiettivi e i dettagli che riguardano la progettazione di una soluzione orientata ai principi del paradigma Zero Trust, integrando meccanismi di autenticazione avanzati che eliminano la dipendenza dalle password.

4.1 Obiettivi

Lo scopo principale di questo progetto è quello di evidenziare i limiti dei modelli di rete industriali tradizionali e proporre una soluzione sicura che tenga in considerazione le specificità di un ambiente OT e i vincoli operativi ad esso associati. I principali obiettivi su cui si concentra l'attenzione sono:

- **Protezione delle comunicazioni**: impedire la possibilità di intercettare o manipolare traffico in chiaro all'interno della rete. Risulta necessario implementare meccanismi in grado di fornire un canale sicuro, garantendo la confidenzialità e l'integrità dei dati trasmessi. In particolare, ogni scambio di informazioni tra due entità deve essere cifrato tramite protocolli sicuri ed efficienti.
- **Segmentazione**: limitare le comunicazioni esclusivamente tra entità esplicitamente autorizzate, riducendo significativamente le possibilità di movimento laterale. La soluzione deve quindi prevedere un approccio basato sulla microsegmentazione, suddividendo l'infrastruttura in segmenti logici isolati, indipendentemente dalla topologia fisica della rete, e applicando politiche di accesso granulari basate sulle identità dei componenti coinvolti, piuttosto che sulla loro posizione all'interno del sistema.
- **Riduzione dell'esposizione dei servizi**: adottare un modello in cui i servizi non siano direttamente esposti sulla rete, ma risultino raggiungibili esclusivamente da entità esplicitamente autorizzate. Questo permette di ridurre significativamente la superficie di attacco e di limitare l'esposizione a eventuali vulnerabilità dei servizi.

- **Autenticazione dei dispositivi**: garantire l'autenticità dei nodi che operano all'interno della rete, assegnando un'identità univoca a ciascuna entità e integrando meccanismi in grado di verificarne la legittimità prima di consentire qualsiasi interazione con gli altri componenti del sistema.

- **Autenticazione degli utenti**: introdurre meccanismi di autenticazione che non prevedano l'utilizzo di credenziali statiche come password o PIN. Il processo deve essere strettamente legato all'identità dell'utente e supportare meccanismi di verifica robusti, garantendo l'accesso ai servizi esclusivamente a entità legittime. Un ulteriore requisito è quello di semplificare la fase di autenticazione, evitando soluzioni che introducano complessità eccessiva per gli operatori.

- **Mantenimento delle prestazioni**: assicurare che l'introduzione di meccanismi di sicurezza non comporti un degrado delle prestazioni del sistema. In particolare, le misure adottate devono essere scalabili e non introdurre overhead significativo, anche in presenza di un numero elevato di dispositivi e comunicazioni simultanee.

Gli obiettivi definiti costituiscono le linee guida utilizzate per la progettazione della soluzione proposta. Nei prossimi capitoli verrà descritta l'architettura del sistema, evidenziando come le scelte progettuali adottate consentano di soddisfare i requisiti individuati e di superare le criticità evidenziate precedentemente.

4.2 Architettura di Base

La progettazione prende avvio dalla definizione di un'architettura di riferimento che descrive il comportamento del sistema in assenza di meccanismi di sicurezza. L'obiettivo è evidenziare in modo chiaro il ruolo dei componenti del sistema e le modalità con cui interagiscono tra loro. Il risultato di questo passaggio costituisce la base sulla quale verranno successivamente introdotte le misure di protezione, elemento centrale del presente lavoro. Nella definizione dell'infrastruttura sono stati introdotti elementi che riflettono in modo realistico le caratteristiche tipiche di un ambiente OT:

- **Programmable Logic Controller (PLC)**: rappresentano le unità di controllo della linea e sono responsabili del coordinamento delle attività tra i diversi componenti del sistema. Un PLC esegue la logica che guida il processo produttivo, gestendo il flusso delle operazioni e sincronizzando le azioni dei dispositivi collegati.

- **Bracci meccanici**: rappresentano i principali attuatori della linea e sono responsabili delle operazioni di assemblaggio e manipolazione dei materiali lungo il processo produttivo. Ogni braccio è associato ad un PLC ed effettua *polling* per determinare i movimenti da compiere. In particolare, vengono richiesti ciclicamente i valori di determinate variabili, i quali corrispondono a specifiche azioni.

- **Nastri trasportatori**: rappresentano il sistema di movimentazione dei materiali lungo la linea produttiva. Consentono il trasferimento degli oggetti tra le diverse stazioni operative, ovvero i bracci meccanici. Anche in questo caso ogni nastro è associato ad un PLC, dal quale ottiene le informazioni necessarie per il controllo del movimento, come l'avvio, l'arresto e la velocità di avanzamento. Il loro funzionamento risulta strettamente coordinato con le operazioni dei bracci robotici.

- **Human Machine Interface (HMI)**: rappresentano le interfacce attraverso le quali gli operatori possono interagire direttamente con il sistema. Ogni HMI consente di visualizzare lo stato di un PLC e di interagire con esso, influenzando indirettamente il comportamento dei dispositivi associati. In questo modo è possibile effettuare il controllo manuale di specifiche operazioni.

- **Supervisory Control and Data Acquisition (SCADA)**: rappresenta il sistema di supervisione della linea produttiva, consentendo il monitoraggio centralizzato dei dispositivi e dello stato complessivo del processo. Attraverso un'interfaccia grafica, lo SCADA permette agli operatori di visualizzare in tempo reale i dati provenienti dai PLC e di interagire con essi, facilitando l'analisi e il controllo delle operazioni.

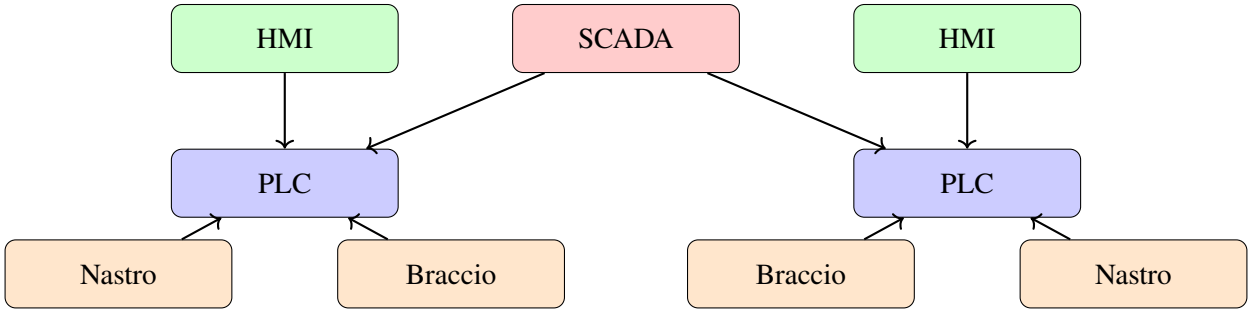

Figura 4.1: Architettura di base del sistema industriale

4.3 Architettura OpenZiti

L'introduzione di OpenZiti consente di superare il modello di comunicazione tradizionale basato su connettività IP diretta. Tale approccio viene sostituito da un paradigma di accesso fondato su identità, servizi logici e politiche di autorizzazione. In questa architettura, i componenti del sistema non risultano più reciprocamente raggiungibili per il solo fatto di appartenere alla medesima rete, ma possono comunicare esclusivamente attraverso l'overlay Zero Trust, secondo regole definite esplicitamente. In un contesto industriale, l'introduzione di nuovi meccanismi di sicurezza deve necessariamente confrontarsi con la presenza di infrastrutture esistenti, spesso caratterizzate da stringenti vincoli operativi e da una limitata possibilità di modifica. Poiché il presente lavoro mira a rappresentare in modo quanto più fedele possibile uno scenario industriale realistico, la progettazione è stata condotta assumendo un contesto di tipo *brownfield*. Tale scelta impone il vincolo di non poter modificare in modo significativo i componenti definiti nell'architettura di base. Di conseguenza, si è fatto ricorso alle funzionalità offerte da OpenZiti per rendere accessibili i servizi esistenti attraverso l'overlay, mediante meccanismi di proxying e tunneling configurati direttamente sui nodi della rete. Questo approccio consente di evitare l'integrazione diretta degli SDK Ziti, che comporterebbe interventi invasivi sul software esistente. I componenti del modello OpenZiti sono già stati descritti nella Sezione 2.3.2; di seguito si procederà pertanto a elencare gli elementi introdotti, senza approfondirne ulteriormente il funzionamento.

L'infrastruttura OpenZiti adottata prevede la presenza di un *controller* e di un *edge router*, ovvero una configurazione specifica del router Ziti in grado di integrarsi alla *mesh network* dell'overlay e

contemporaneamente fungere da terminatore per uno o più servizi Ziti. Ciò costituisce l'architettura minima necessaria alla creazione di una rete OpenZiti funzionante.
Tuttavia, tale configurazione non è di per sé sufficiente a consentire alle entità del sistema di partecipare attivamente all'infrastruttura sicura. A tal fine, si ricorre ai *tunneler*, componenti software installati direttamente sui nodi della rete in grado di intercettare il traffico in uscita e instradarlo verso il router Ziti, nonché di gestire il traffico proveniente dall'overlay verso i processi locali, il tutto in modo completamente trasparente alle applicazioni.

Ciascun *tunneler* è associato a una specifica identità Ziti, che ne determina i permessi all'interno dell'overlay. Nel contesto della linea industriale considerata, si distinguono due funzioni principali delle identità:

- **Identità client**: associate ai componenti che necessitano di accedere a servizi, quali bracci meccanici, nastri trasportatori, interfacce HMI e sistema SCADA. Tali entità richiedono la connessione a determinati nodi per ottenere informazioni o inviare comandi.
- **Identità di servizio**: associate ai componenti che espongono funzionalità all'interno della rete, in particolare ai PLC. In questo caso, il *tunneler* consente l'accesso ai servizi offerti dal dispositivo attraverso l'overlay tramite il router, senza la necessità di esporre porte.

Questa distinzione consente di modellare esplicitamente le relazioni tra i diversi componenti del sistema e di applicare politiche di accesso granulari, in cui le entità possono interagire esclusivamente se autorizzate. In OpenZiti, il controllo degli accessi è mediato attraverso il concetto di *servizio*, un'entità logica che astrae una risorsa reale, consentendo di disaccoppiare completamente l'accesso dalla sua collocazione fisica o topologica. I servizi vengono associati alle identità di servizio, che ne rendono possibile l'esposizione all'interno dell'overlay.

La gestione delle autorizzazioni è determinata dalle *politiche*, che, se configurate correttamente, consentono di implementare il principio del minimo privilegio, garantendo che ciascuna entità possa accedere esclusivamente alle risorse strettamente necessarie al proprio funzionamento. In particolare, si distinguono due tipologie di politiche:

- **Bind policy**: definiscono quali identità sono autorizzate a esporre un determinato servizio all'interno dell'overlay. Tali politiche vengono quindi associate alle identità di servizio, che nel caso in esame consentono ai PLC di pubblicare i rispettivi servizi.

- **Dial policy**: definiscono quali identità client sono autorizzate a instaurare una connessione verso uno specifico servizio. Nel sistema progettato, queste politiche vengono utilizzate per limitare l'accesso ai PLC esclusivamente ai componenti ad esso associati. Nello specifico, a PLC1 potranno accedere unicamente Braccio1, Nastro1, HMI1 e SCADA.

L'adozione combinata di tali politiche consente di separare in modo netto chi può esporre una risorsa da chi può utilizzarla, in piena aderenza al principio del minimo privilegio, nel quale ciascuna entità può svolgere esclusivamente il ruolo previsto all'interno del processo industriale.

L'esposizione e l'utilizzo dei servizi all'interno dell'overlay sono inoltre definiti attraverso specifiche *configurazioni*, che descrivono le modalità di interazione tra client e risorse.
Nel caso in esame, sono state utilizzate principalmente due tipologie di configurazione: *host.v1* e *intercept.v1*. La prima consente di definire le caratteristiche del servizio lato erogatore, specificando indirizzo e porta locali presso cui il processo reale è disponibile, tipicamente in ambito locale al nodo. La seconda, invece, definisce le modalità con cui il traffico viene intercettato lato client, associando alla risorsa in questione un hostname riconosciuto all'interno della rete OpenZiti.
Ciò permette ai client di accedere alle risorse utilizzando nomi logici, senza alcun riferimento diretto a indirizzi IP o porte della rete sottostante. Questo meccanismo realizza un completo disaccoppiamento tra l'identità del servizio e la sua collocazione fisica, eliminando la dipendenza dalla topologia di rete e rafforzando il modello di sicurezza Zero Trust.

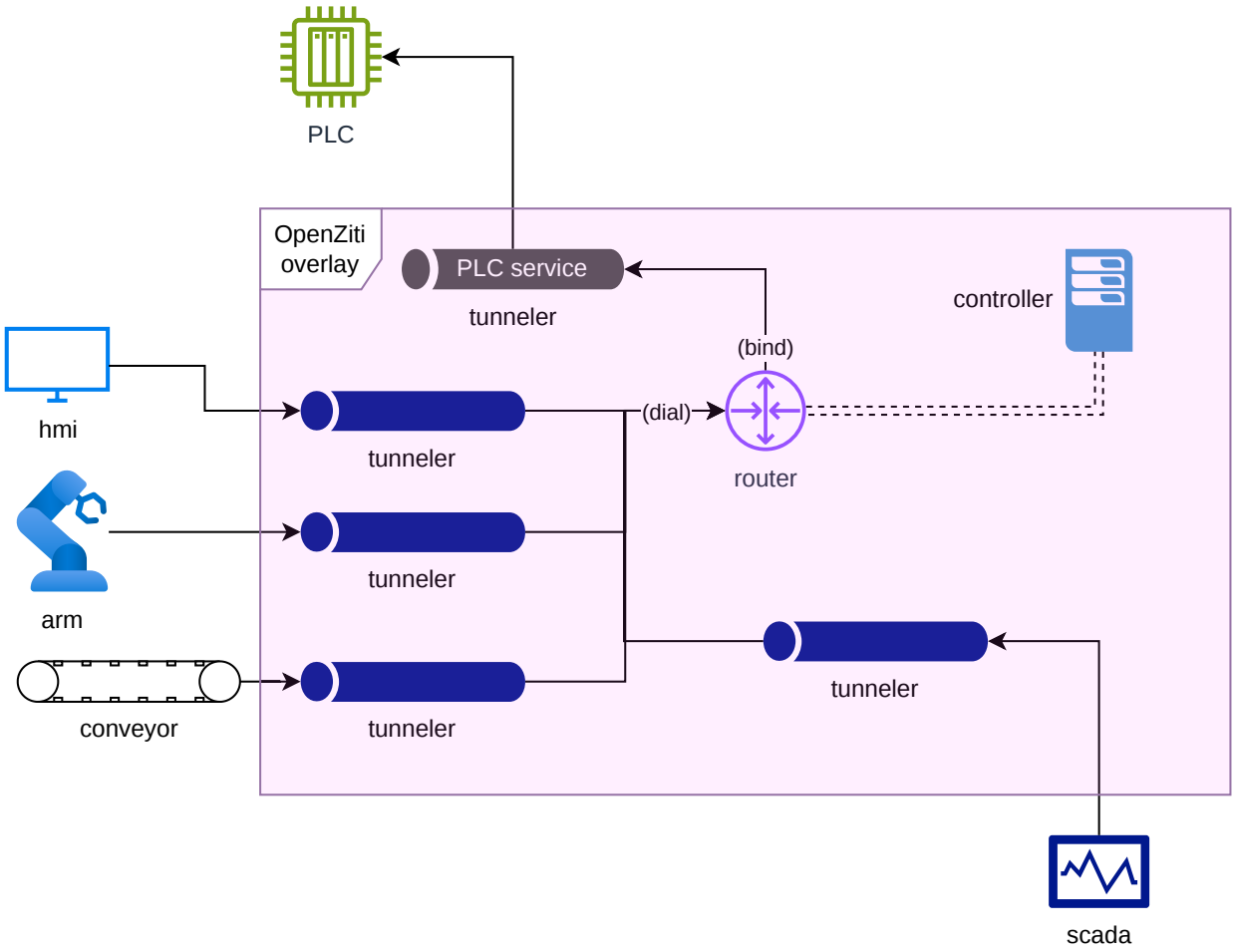

Figura 4.2: Architettura Zero Trust basata su OpenZiti

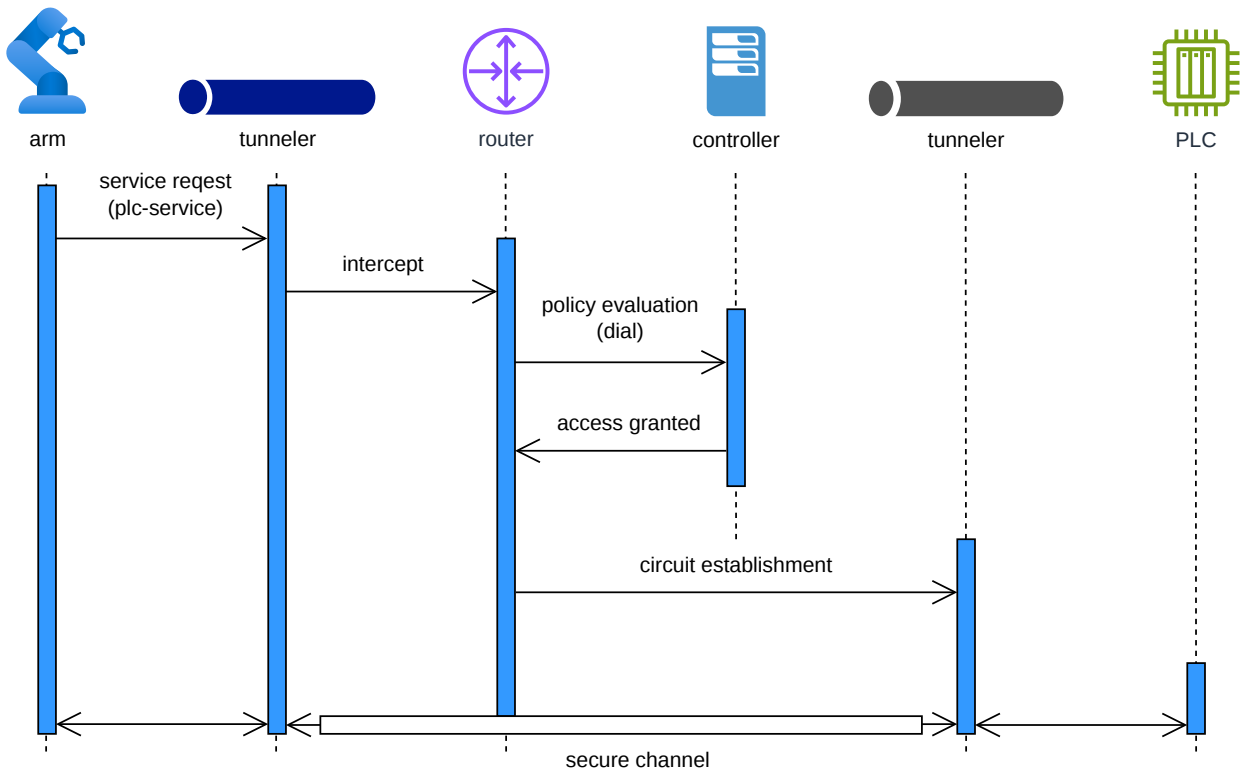

Figura 4.3: Flusso di accesso a un servizio nell'overlay OpenZiti

L'introduzione dell'overlay OpenZiti consente di soddisfare una parte significativa degli obiettivi di sicurezza definiti, in particolare:

- **Protezione delle comunicazioni**: tutte le comunicazioni tra le entità del sistema avvengono attraverso circuiti sicuri instaurati all'interno dell'overlay. Ciò garantisce la cifratura del traffico end-to-end, impedendo l'intercettazione e la manipolazione dei dati, assicurando quindi confidenzialità e integrità delle informazioni scambiate.
- **Segmentazione**: l'adozione di un modello basato su servizi logici e politiche di accesso consente di realizzare una microsegmentazione indipendente dalla topologia fisica della rete. Le comunicazioni risultano limitate esclusivamente alle entità esplicitamente autorizzate, riducendo significativamente le possibilità di movimento laterale all'interno dell'infrastruttura.
- **Riduzione dell'esposizione dei servizi**: non è più necessario esporre i servizi direttamente sulla rete sottostante. Le risorse risultano accessibili attraverso l'overlay, previa esplicita autorizzazione, con una conseguente riduzione della superficie di attacco.

- **Autenticazione dei dispositivi**: ciascun nodo dell'infrastruttura è associato a un'identità univoca all'interno dell'overlay OpenZiti. L'accesso ai servizi è subordinato alla verifica di tale identità, garantendo che solo dispositivi legittimi possano partecipare alle comunicazioni.
- **Mantenimento delle prestazioni**: l'architettura adottata consente di introdurre i meccanismi di sicurezza senza modificare i servizi esistenti grazie all'utilizzo dei *tunneler*. Inoltre, l'instradamento efficiente del traffico permette di mantenere un livello di prestazioni adeguato anche in presenza di più comunicazioni simultanee. L'infrastruttura risulta inoltre scalabile, consentendo l'introduzione di ulteriori router a supporto di un eventuale incremento di nodi nella rete.

L'obiettivo relativo all'autenticazione degli utenti non è direttamente affrontato da questa soluzione e verrà trattato nella sezione successiva mediante l'introduzione di meccanismi di autenticazione passwordless basati su WebAuthn.

4.4 Autenticazione WebAuthn

Come discusso nella sezione precedente, l'introduzione dell'overlay OpenZiti consente di garantire un elevato livello di sicurezza per quanto riguarda la protezione delle comunicazioni, la segmentazione dell'infrastruttura e l'autenticazione dei dispositivi. Tuttavia, tale soluzione non affronta direttamente il problema dell'autenticazione degli utenti, che rappresenta un aspetto altrettanto critico in ambienti industriali.

I meccanismi di autenticazione tradizionali basati su credenziali statiche, quali password o PIN, presentano infatti numerose limitazioni e risultano poco adatti a contesti industriali, nei quali è necessario garantire accessi rapidi, affidabili e con un ridotto margine di errore umano.

Per rispondere a tali criticità, nel presente lavoro si propone l'adozione di un approccio passwordless basato su WebAuthn, in cui l'identità dell'utente viene legata al possesso di un autenticatore fisico o software, eliminando la necessità di gestire credenziali segrete.

L'integrazione di WebAuthn con l'infrastruttura OpenZiti consente di estendere il paradigma Zero Trust anche a livello utente, permettendo di ottenere un modello di sicurezza completo in cui

ogni entità coinvolta nel sistema, sia essa umana o automatizzata, è soggetta a meccanismi di autenticazione e autorizzazione espliciti.

L'adozione di WebAuthn all'interno dell'infrastruttura proposta richiede l'introduzione di componenti specifici per la gestione delle identità degli utenti e per il controllo degli accessi alle risorse. Si rende quindi necessario lo sviluppo di un'infrastruttura applicativa dedicata.
A tal fine, sono stati introdotti tre elementi principali: un componente per la gestione delle identità utente, un componente per il controllo degli accessi alle interfacce esposte e un sistema di persistenza per la memorizzazione delle credenziali WebAuthn. Nei paragrafi seguenti vengono descritti nel dettaglio il ruolo e il funzionamento dei componenti introdotti.

4.4.1 Database

La gestione delle credenziali WebAuthn e delle informazioni degli utenti richiede la presenza di un sistema di persistenza in grado di memorizzare in modo strutturato i dati necessari al processo di autenticazione.

In particolare, per ciascun utente vengono memorizzate le seguenti informazioni:

- **id**: identificativo univoco dell'utente all'interno del sistema.
- **username**: nome utente utilizzato per visualizzare l'operatore all'interno del sistema.
- **name**: nome proprio dell'utente, utilizzato a fini informativi.
- **credential_id**: identificativo della chiave privata WebAuthn associata all'utente.
- **credential_public_key**: chiave pubblica utilizzata per la verifica della firma digitale durante il processo di autenticazione.
- **role**: ruolo associato all'utente, solo l'amministratore è in grado di accedere allo *UserHandler*.

User		
PK	**id**	**Integer**
	username	String(80)
	name	String(120)
	credential_public_key	LargeBinary
	current_sign_count	Integer
	credential_id	LargeBinary
	role	String(5)

Figura 4.4: Schema Entity-Relationship del database

È importante sottolineare come, coerentemente con il modello proposto, il sistema non memorizzi alcuna credenziale segreta, ma esclusivamente dati pubblici necessari alla verifica delle firme digitali durante il processo di autenticazione. Questo approccio riduce significativamente i rischi associati alla compromissione del database.

4.4.2 UserHandler: gestione degli utenti

Il componente *UserHandler* rappresenta il modulo responsabile della gestione delle identità utente. Esso fornisce le funzionalità necessarie alla registrazione e rimozione degli utenti, oltre alla gestione delle credenziali associate ai meccanismi di autenticazione WebAuthn.

In particolare, durante la fase di registrazione, il componente *UserHandler* interagisce con il client per avviare il processo di creazione delle credenziali WebAuthn, memorizzando nel database le informazioni necessarie alla successiva verifica delle autenticazioni.

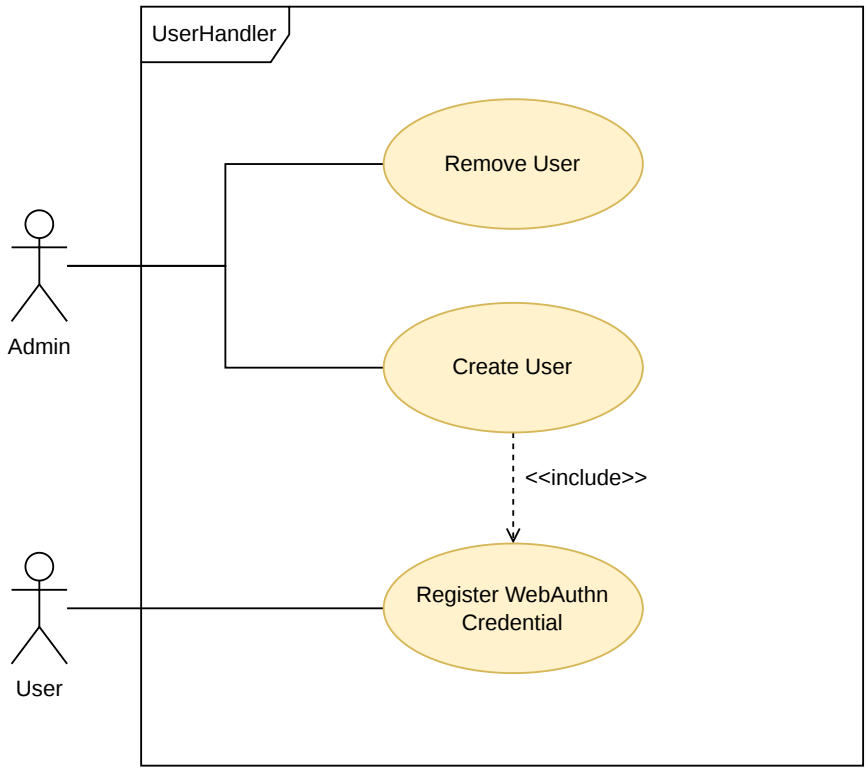

Figura 4.5: Diagramma dei casi d'uso del componente *UserHandler*

Il componente svolge quindi un ruolo centrale nella gestione delle identità digitali degli operatori, costituendo il punto di riferimento per l'amministrazione degli utenti.

4.4.3 LoginGateway: controllo degli accessi

Il componente *LoginGateway* rappresenta il punto di ingresso per tutte le interfacce web esposte. Il suo compito principale è quello di garantire che ogni richiesta proveniente da un utente sia sottoposta a un processo di autenticazione prima di poter accedere alle risorse protette.
Tale approccio consente di applicare il principio *never trust, always verify* anche alle interazioni umane con il sistema, estendendo il paradigma Zero Trust oltre il livello di rete.
Il *LoginGateway* assume pertanto il ruolo di punto di enforcement delle politiche di accesso lato utente.

L'introduzione di questo meccanismo consente di soddisfare gli obiettivi di sicurezza non ancora coperti:

- **Autenticazione degli utenti**: l'accesso alle risorse è subordinato alla verifica esplicita dell'identità dell'utente mediante credenziali crittografiche. Ciò garantisce che solo utenti legittimi possano interagire con il sistema. L'adozione di un approccio passwordless consente di eliminare l'utilizzo di password o PIN, riducendo i rischi associati al furto di credenziali, e semplifica il processo di autenticazione per gli operatori, riducendo la probabilità di errori legati alla gestione delle credenziali.
- **Mantenimento delle prestazioni**: l'architettura introdotta prevede un numero limitato di componenti aggiuntivi e non comporta un aumento significativo del traffico di rete, mantenendo contenuto l'overhead complessivo del sistema.

4.5 Integrazione su architettura complessa

Nelle sezioni precedenti sono stati introdotti i meccanismi di sicurezza adottati per la protezione dell'infrastruttura, con particolare riferimento all'utilizzo di dell'overlay OpenZiti per la gestione delle comunicazioni tra i dispositivi e all'impiego di WebAuthn per l'autenticazione degli utenti.
Sebbene tali componenti siano stati analizzati separatamente, è la loro integrazione a consentire la realizzazione di un'architettura di sicurezza completa, in grado di garantire sia la protezione del traffico sia il controllo degli accessi alle risorse.
In questa sezione vengono descritte le modalità con cui i due approcci vengono combinati all'interno del sistema proposto, definendo un modello in accordo con il paradigma Zero Trust.

4.5.1 Reverse Proxy

L'integrazione dei meccanismi di autenticazione degli utenti all'interno dell'infrastruttura proposta deve confrontarsi con il vincolo, già evidenziato, di operare in un contesto di tipo *brownfield*, nel quale non è possibile alterare in modo significativo i servizi esistenti.
In particolare, le interfacce dei componenti del sistema, quali SCADA e HMI, risultano già implementate e non progettate per integrare nativamente meccanismi di autenticazione avanzati come WebAuthn. Ciò rende necessario individuare una soluzione in grado di introdurre un controllo degli accessi senza intervenire direttamente sulla logica applicativa dei servizi.

A tal fine, è stato introdotto un componente di tipo reverse proxy, incaricato di intercettare tutte le richieste provenienti dagli utenti per sottoporle a un controllo di autenticazione tramite il *LoginGateway* prima di consentire l'accesso alle risorse protette.

Un reverse proxy è un elemento intermedio che si pone tra i client e i servizi applicativi, ricevendo le richieste in ingresso e inoltrandole al servizio di destinazione. Di fatto rappresenta il punto di accesso alle risorse, mascherandone la reale collocazione e centralizzando la gestione delle richieste. Dal punto di vista operativo, quindi, l'utente non interagisce direttamente con la risorsa finale, ma invia le proprie richieste al reverse proxy, il quale le valuta e le inoltra al servizio appropriato. Le risposte generate vengono poi restituite al client attraverso lo stesso componente, rendendo trasparente l'intermediazione.

Nel contesto dell'architettura proposta, il reverse proxy viene installato su ogni nodo che fornisce un'interfaccia accessibile da un operatore. Per ogni richiesta, esso verifica la presenza di una sessione valida tramite il *LoginGateway*; in caso contrario, l'utente viene reindirizzato alla schermata di login. Una volta completata con successo l'autenticazione, le richieste vengono inoltrate al servizio del nodo.

In questo modo, viene imposto un flusso di accesso controllato, in cui ogni utente è obbligato a completare il processo di autenticazione prima di poter interagire con i servizi del sistema. Tale approccio consente di estendere questi meccanismi di sicurezza anche a componenti legacy, risultando pienamente coerente con i vincoli operativi del contesto industriale considerato.

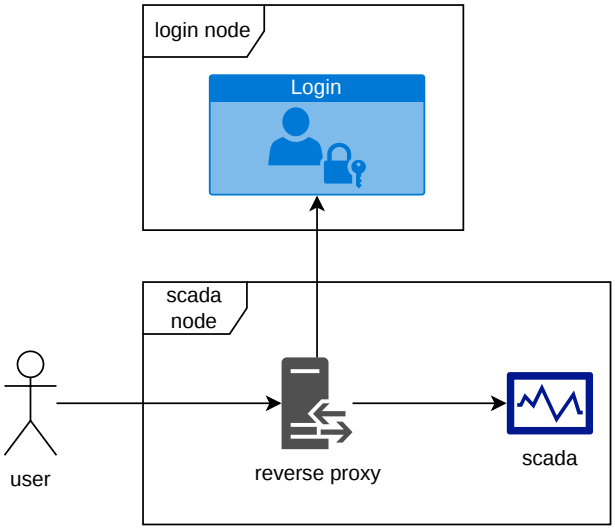

Figura 4.6: Diagramma architettura reverse proxy + *LoginGateway*

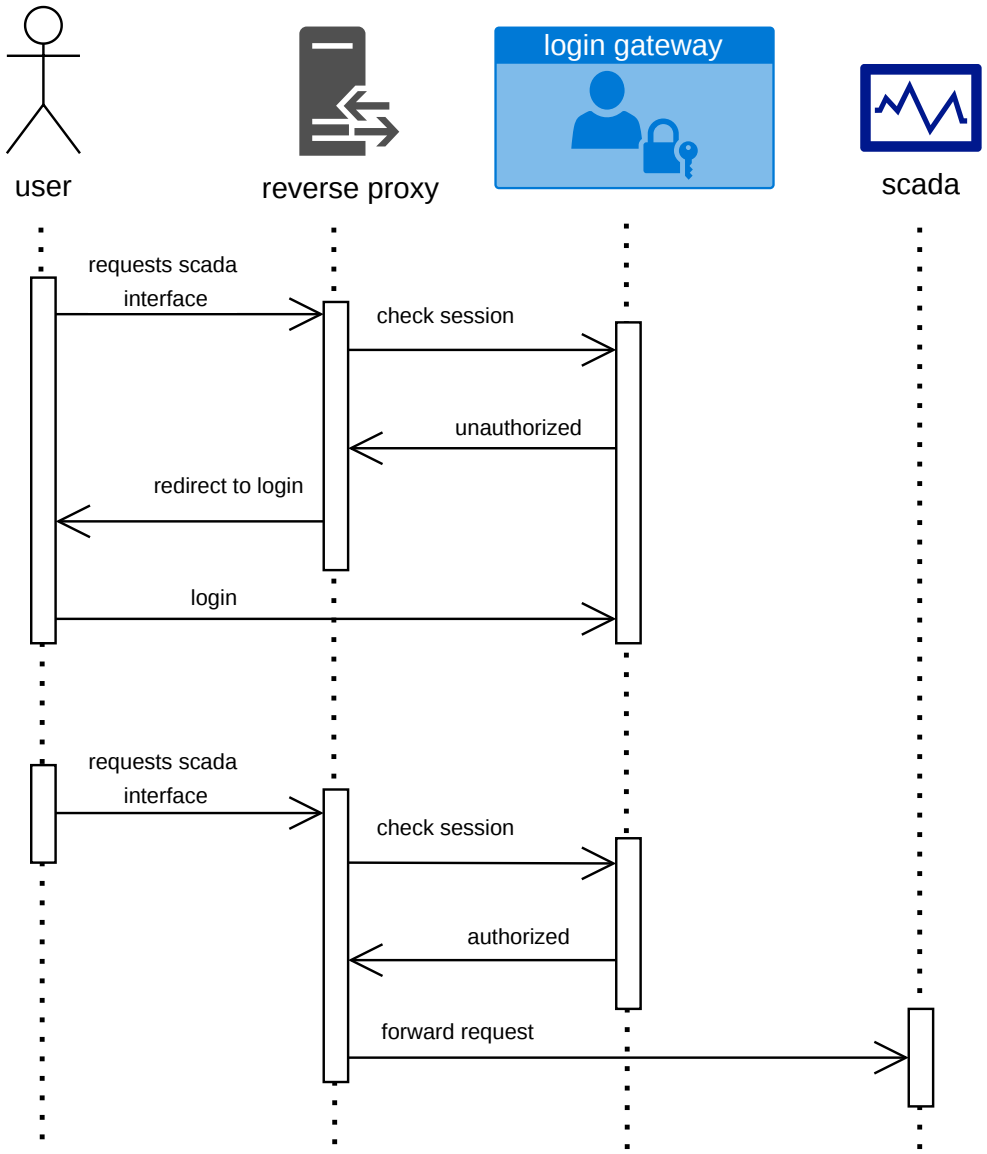

Figura 4.7: Diagramma sequenza reverse proxy + *LoginGateway*

4.5.2 Integrazione WebAuthn e OpenZiti

L'introduzione dei componenti applicativi necessari al supporto di WebAuthn richiede la loro integrazione all'interno dell'overlay OpenZiti, al fine di mantenere coerenza con il modello di sicurezza adottato. Anche tali componenti, infatti, non devono risultare direttamente esposti o raggiungibili sulla rete sottostante, ma devono essere accessibili esclusivamente tramite identità, servizi logici e politiche esplicite.

In analogia con quanto definito per i componenti OT, ciascun elemento introdotto viene modellato all'interno dell'overlay mediante una propria identità OpenZiti e un tunneler associato. In particolare, vengono definite identità dedicate per il *LoginGateway*, per il *UserHandler* e per il database.
Il reverse proxy, invece, non è direttamente integrato nell'overlay OpenZiti, in quanto è previsto venga eseguito localmente sui nodi che ospitano le interfacce da proteggere.

A partire da tali identità, viene definito un servizio OpenZiti per il database, in quanto esso deve risultare accessibile dal *LoginGateway* e dal *UserHandler*.
L'esposizione del servizio è regolata tramite una politica di tipo *Bind*, che autorizza l'identità associata al database a pubblicarne le funzionalità all'interno dell'overlay. Parallelamente, vengono definite politiche di tipo *Dial* per limitare le comunicazioni verso il database esclusivamente alle identità associate a *LoginGateway* e *UserHandler*.
Le modalità di esposizione e di accesso sono inoltre definite tramite configurazioni OpenZiti di tipo *host.v1* e *intercept.v1*. La prima specifica l'endpoint locale su cui il database risulta disponibile, mentre la seconda associa il servizio a un hostname logico all'interno dell'overlay, consentendo ai nodi autorizzati di accedervi senza alcun riferimento diretto alla rete IP sottostante.

Infine, l'accesso alle interfacce a livello di rete viene limitato mediante opportune regole di firewall, che consentono esclusivamente la comunicazione con i componenti esposti, quali il reverse proxy, il *LoginGateway* e il *UserHandler*, quest'ultimo dotato di un proprio meccanismo di login interno. In questo modo, si impedisce, qualsiasi accesso diretto ai servizi dell'infrastruttura OT, garantendo che ogni interazione avvenga esclusivamente attraverso i meccanismi di controllo definiti nell'overlay e nel sistema di autenticazione.

4.5.3 Flusso di accesso alle risorse

L'integrazione tra il reverse proxy, il sistema di autenticazione basato su WebAuthn e l'overlay OpenZiti consente di definire un flusso di accesso alle risorse completamente controllato, nel quale ogni richiesta è sottoposta a verifiche esplicite sia a livello utente sia a livello di servizio.
Quando un utente tenta di accedere a una delle interfacce esposte, ad esempio al sistema SCADA, la richiesta viene intercettata dal reverse proxy, che rappresenta il punto di ingresso unico alle risorse.

In questa fase, il proxy non inoltra immediatamente la richiesta al servizio di destinazione, ma verifica preliminarmente la presenza di una sessione utente valida tramite il *LoginGateway*.
In assenza di una sessione valida, il proxy impedisce l'accesso alla risorsa e reindirizza l'utente verso l'interfaccia di autenticazione. Il processo di login avviene tramite WebAuthn, che consente di verificare l'identità dell'utente mediante credenziali crittografiche associate a un autenticatore.
Una volta completata con successo l'autenticazione, l'utente ottiene una sessione valida e può ripetere la richiesta iniziale. A questo punto, il reverse proxy consente il proseguimento del flusso e inoltra la richiesta verso l'interfaccia SCADA.
Le interazioni tra il servizio SCADA e i PLC avvengono interamente all'interno dell'overlay OpenZiti. Ogni scambio è autenticato e cifrato, garantendo la confidenzialità e l'integrità dei dati, riducendo significativamente la possibilità di intercettazione o manipolazione del traffico.

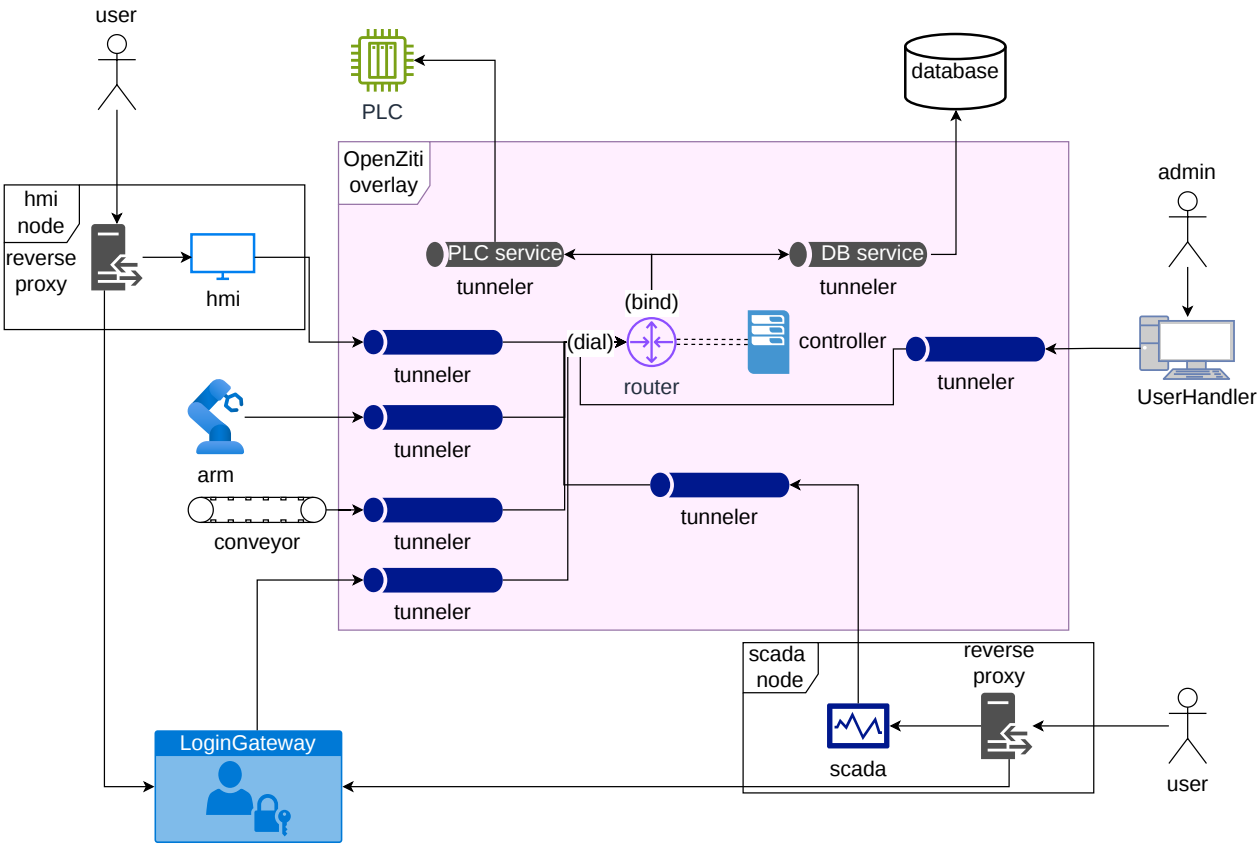

Figura 4.8: Architettura OpenZiti + WebAuthn

In questo modo, ogni accesso alle risorse risulta subordinato a un duplice controllo: l'autenticazione

dell'utente, gestita dal *LoginGateway*, e l'autorizzazione alla comunicazione tra i servizi, garantita dall'overlay OpenZiti. Questo approccio consente di estendere il paradigma Zero Trust a tutti i livelli del sistema, dalla gestione degli utenti ai servizi OT interni.

Inoltre, l'architettura proposta consente di introdurre tali meccanismi di sicurezza mantenendo un overhead limitato, garantendo prestazioni adeguate al corretto funzionamento dell'infrastruttura.

5. Implementazione

Il presente capitolo descrive in dettaglio l'implementazione dell'architettura proposta, evidenziando le soluzioni adottate, le tecnologie impiegate e le modalità di integrazione dei diversi componenti del sistema.

L'infrastruttura è stata articolata in due ambienti distinti ma complementari:

- **Testbed virtuale**: soluzione completamente simulata, che consente una rapida configurazione, elevata replicabilità degli esperimenti e semplicità di integrazione dei componenti;
- **Testbed fisico**: composto da dispositivi hardware reali, con l'obiettivo di emulare una linea pilota industriale e dimostrare l'applicabilità della soluzione in un contesto operativo realistico.

Entrambi gli ambienti implementano la medesima architettura logica e sono stati configurati secondo due distinti scenari operativi: un ambiente privo di meccanismi di sicurezza, finalizzato a evidenziare le vulnerabilità tipiche delle architetture OT tradizionali, e un ambiente protetto, in cui vengono adottate le soluzioni progettate al fine di garantire la sicurezza del sistema.
Il codice sorgente dell'intero progetto è disponibile pubblicamente su GitHub[1].

5.1 Comunicazione tra i componenti OT

La comunicazione tra i diversi componenti del sistema OT è realizzata mediante il protocollo Modbus TCP [10], ampiamente utilizzato in ambito industriale per lo scambio di dati tra dispositivi di controllo e sistemi di supervisione.
Modbus TCP rappresenta un'estensione del protocollo Modbus [9] che utilizza TCP/IP come livello di trasporto, consentendo l'integrazione delle reti industriali con infrastrutture di comunicazione standard. Il protocollo segue un modello client-server (storicamente master-slave), in cui un dispositivo client invia richieste a uno o più dispositivi server per la lettura o la scrittura di registri.

[1] `https://github.com/Tragyt/ICS`

I dati sono organizzati in registri, che rappresentano le variabili di processo e lo stato dei dispositivi. Le operazioni principali consistono nella lettura e scrittura di tali registri, permettendo l'interazione tra diversi componenti del sistema.
I registri Modbus sono tipicamente suddivisi in diverse categorie, tra cui coil (uscite discrete), input discreti, holding register e input register, ciascuna utilizzata per rappresentare specifiche tipologie di dati.
Nella soluzione proposta, Modbus TCP è utilizzato come meccanismo di comunicazione tra i PLC, che svolgono il ruolo di server, e i client, ovvero i sistemi di supervisione (HMI e SCADA) e i dispositivi di campo, quali bracci robotici e nastri trasportatori. Tale configurazione risulta coerente con le architetture OT reali.
Tale configurazione introduce inoltre implicazioni rilevanti dal punto di vista della sicurezza, in quanto il protocollo Modbus TCP non prevede meccanismi nativi di autenticazione o cifratura, rendendolo particolarmente esposto ad attacchi in assenza di adeguate contromisure. Tale caratteristica rappresenta un ulteriore elemento che motiva l'adozione di un'architettura basata su principi Zero Trust.

5.2 Tecnologie utilizzate

La realizzazione del sistema proposto si basa sull'integrazione di tecnologie eterogenee, appartenenti al dominio dei sistemi OT e a quello delle infrastrutture IT. In particolare, l'architettura utilizza OpenZiti e WebAuthn come soluzioni per la sicurezza, ampiamente descritte nel Capitolo 2.
Poiché le medesime tecnologie sono impiegate sia nel testbed virtuale sia in quello fisico, esse vengono presentate congiuntamente nella presente sezione, al fine di evitare ridondanze e fornire una visione unitaria dello stack tecnologico adottato.

Di seguito sono descritte le tecnologie principali e maggiormente rilevanti per l'architettura proposta, mentre ulteriori componenti saranno introdotti e discussi nel dettaglio nelle sezioni successive dedicate all'implementazione:

- **Docker**: piattaforma di containerizzazione che consente di eseguire applicazioni all'interno di ambienti isolati, denominati container. Ciascun container include il codice applicativo,

le librerie e le dipendenze necessarie al suo funzionamento, garantendo coerenza tra diversi ambienti di esecuzione. A differenza delle tradizionali soluzioni di virtualizzazione basate su macchine virtuali, i container condividono il kernel del sistema operativo host, risultando più leggeri e permettendo tempi di avvio significativamente ridotti. Questa caratteristica consente di eseguire un elevato numero di servizi su una singola macchina, mantenendo al contempo un buon livello di isolamento tra i diversi componenti. Docker si basa sul concetto di immagini, che rappresentano template immutabili utilizzati per creare gli ambienti containerizzati, e su un sistema di stratificazione (layering) che consente di ottimizzare lo spazio occupato e il riutilizzo di componenti comuni. La costruzione delle immagini è definita tramite file di configurazione denominati Dockerfile, che descrivono in maniera dichiarativa le operazioni necessarie per l'allestimento dell'ambiente di esecuzione, come l'installazione delle dipendenze, la configurazione dei servizi e la definizione dei comandi di avvio. Tali caratteristiche rendono Docker particolarmente adatto alla realizzazione del sistema proposto, che richiede l'esecuzione simultanea di un numero elevato di nodi, in particolare nell'ambiente sicuro.

- **Docker Compose**: strumento per l'orchestrazione di applicazioni multi-container, che consente di definire e gestire l'esecuzione coordinata di più servizi Docker attraverso file di configurazione dichiarativi. In particolare, Docker Compose utilizza file in formato YAML per descrivere i servizi, le reti e i volumi necessari al funzionamento del sistema, permettendo di avviare l'intera infrastruttura con un singolo comando. Questo approccio consente di semplificare la gestione di sistemi complessi composti da numerosi componenti interdipendenti, garantendo coerenza nella configurazione e facilitando la replicabilità dell'ambiente. Inoltre, Docker Compose permette di definire in modo esplicito le relazioni tra i servizi, incluse le dipendenze e le modalità di comunicazione tra i diversi container. Nel sistema proposto, Docker Compose è utilizzato per orchestrare l'intero ambiente virtuale, includendo i componenti ICS, i servizi applicativi e i meccanismi di sicurezza, consentendo una gestione centralizzata e una rapida riconfigurazione dei diversi scenari sperimentali. Lo strumento si è rivelato efficace anche nel contesto del testbed fisico, permettendo il deployment rapido di determinati servizi su dispositivi distribuiti.

- **OpenPLC** [12]: piattaforma open-source per l'implementazione di controllori logici programmabili (PLC), progettata per l'esecuzione di logiche di controllo industriale in conformità allo standard IEC 61131-3 [11]. Essa consente di sviluppare e testare programmi di automazione utilizzando linguaggi tipici del dominio OT, quali Ladder Diagram (LD), Structured Text (ST) e Function Block Diagram (FBD), definiti dallo standard stesso. OpenPLC può essere eseguito su diverse piattaforme hardware e software, permettendo di simulare il comportamento di PLC reali anche in ambienti virtualizzati. La piattaforma include un runtime che esegue ciclicamente la logica di controllo, interfacciandosi con ingressi e uscite per la gestione dei dispositivi industriali. Oltre al runtime, è inoltre inclusa un'interfaccia web che consente il monitoraggio e la gestione del controllore. Nel sistema proposto, OpenPLC è utilizzato per simulare i PLC della linea, mentre l'interfaccia web assolve il ruolo di Human-Machine Interface (HMI), permettendo l'interazione con le variabili dei controllori e la supervisione del loro stato. L'utilizzo di OpenPLC consente di riprodurre in modo realistico il comportamento dei sistemi OT, mantenendo al contempo la flessibilità necessaria per l'integrazione con l'infrastruttura containerizzata e con i meccanismi di sicurezza introdotti.

- **ScadaBR** [13]: sistema SCADA open-source utilizzato per la supervisione e il monitoraggio di processi industriali. Esso consente la raccolta, la visualizzazione e la gestione dei dati provenienti da dispositivi di campo, quali PLC e sensori, attraverso interfacce grafiche accessibili via web. ScadaBR permette di definire punti dati, configurare dashboard e creare interfacce di supervisione per l'osservazione dello stato del sistema e l'interazione con i processi controllati. Il sistema supporta diversi protocolli di comunicazione industriale, consentendo l'integrazione con dispositivi eterogenei.

- **NGINX** [7]: web server e reverse proxy ad alte prestazioni, ampiamente utilizzato per la gestione del traffico HTTP e l'instradamento delle richieste verso servizi backend. Esso consente di implementare meccanismi di bilanciamento del carico, caching e controllo degli accessi, risultando particolarmente adatto alla realizzazione di architetture distribuite. Nel sistema proposto, NGINX è impiegato come punto di ingresso per l'accesso alle interfacce web dei servizi, svolgendo il ruolo di reverse proxy e di componente di enforcement per le politiche di sicurezza. In particolare, esso consente di proteggere l'accesso alle risorse

attraverso meccanismi di autenticazione delegata, integrandosi con i servizi di autenticazione implementati.

Le tecnologie descritte costituiscono la base per la realizzazione dell'architettura proposta. Nel seguito verranno illustrate le modalità di integrazione di tali componenti nei diversi ambienti sviluppati, evidenziando le scelte implementative adottate.

5.3 Testbed Virtuale

Il testbed virtuale rappresenta l'ambiente principale utilizzato per la progettazione, l'implementazione e la validazione preliminare dell'architettura proposta. Esso consente di simulare in modo controllato il comportamento di un sistema ICS, riproducendo le interazioni tra i componenti principali di un'infrastruttura industriale.
L'adozione di un sistema completamente containerizzato, permette di ottenere un ambiente altamente modulare e facilmente replicabile, facilitando la distribuzione dei servizi e la gestione delle dipendenze. Tale approccio risulta inoltre particolarmente utile anche in ottica di estensione verso l'ambiente fisico, come verrà approfondito nella sezione successiva.
Inoltre, la containerizzazione consente di isolare i singoli componenti del sistema, rendendo possibile l'analisi dettagliata del comportamento di ciascun elemento e l'esecuzione di esperimenti in condizioni controllate.

5.3.1 Architettura ICS

In questa sezione vengono descritti i singoli elementi che compongono l'infrastruttura OT del testbed virtuale, evidenziandone ruolo, modalità di implementazione e interazioni reciproche. Ciascun componente del sistema è realizzato come un servizio containerizzato, al fine di garantire isolamento, modularità e replicabilità. In particolare, saranno analizzati i dispositivi simulati che emulano il comportamento della linea, le interfacce di supervisione e i componenti software necessari al funzionamento complessivo del sistema.

PLC e HMI simulati

PLC e HMI vengono analizzati congiuntamente in quanto, come già descritto nella Sezione 5.1, OpenPLC fornisce sia un runtime per l'esecuzione della logica di controllo sia un'interfaccia web per il monitoraggio dello stato e l'interazione con le variabili del sistema, svolgendo di fatto il ruolo di Human Machine Interface.
Nel testbed virtuale sono state previste quattro istanze OpenPLC, ciascuna eseguita all'interno di un container dedicato. Ogni istanza simula quindi un Programmable Logic Controller con la relativa interfaccia utente.

Logica di controllo La logica di controllo dei PLC è stata implementata mediante lo strumento OpenPLC Editor, un ambiente di sviluppo che supporta i linguaggi definiti dallo standard IEC 61131-3. Tra questi è stato scelto il linguaggio Ladder Logic (LD), in quanto consente di modellare in modo intuitivo sistemi di controllo basati su logica booleana attraverso rappresentazioni grafiche. Il codice definisce il comportamento dei singoli nodi della linea, modellando le transizioni di stato e le interazioni tra i diversi componenti. In particolare, la logica implementa meccanismi di controllo sequenziale, nei quali l'attivazione di specifiche uscite dipende dallo stato corrente del sistema e dai valori degli ingressi.

In particolare, il programma mostrato in Figura 5.1 descrive il funzionamento di un controllore associato a un attuatore verticale, responsabile della gestione della presenza del pezzo e del movimento di discesa e risalita del braccio.

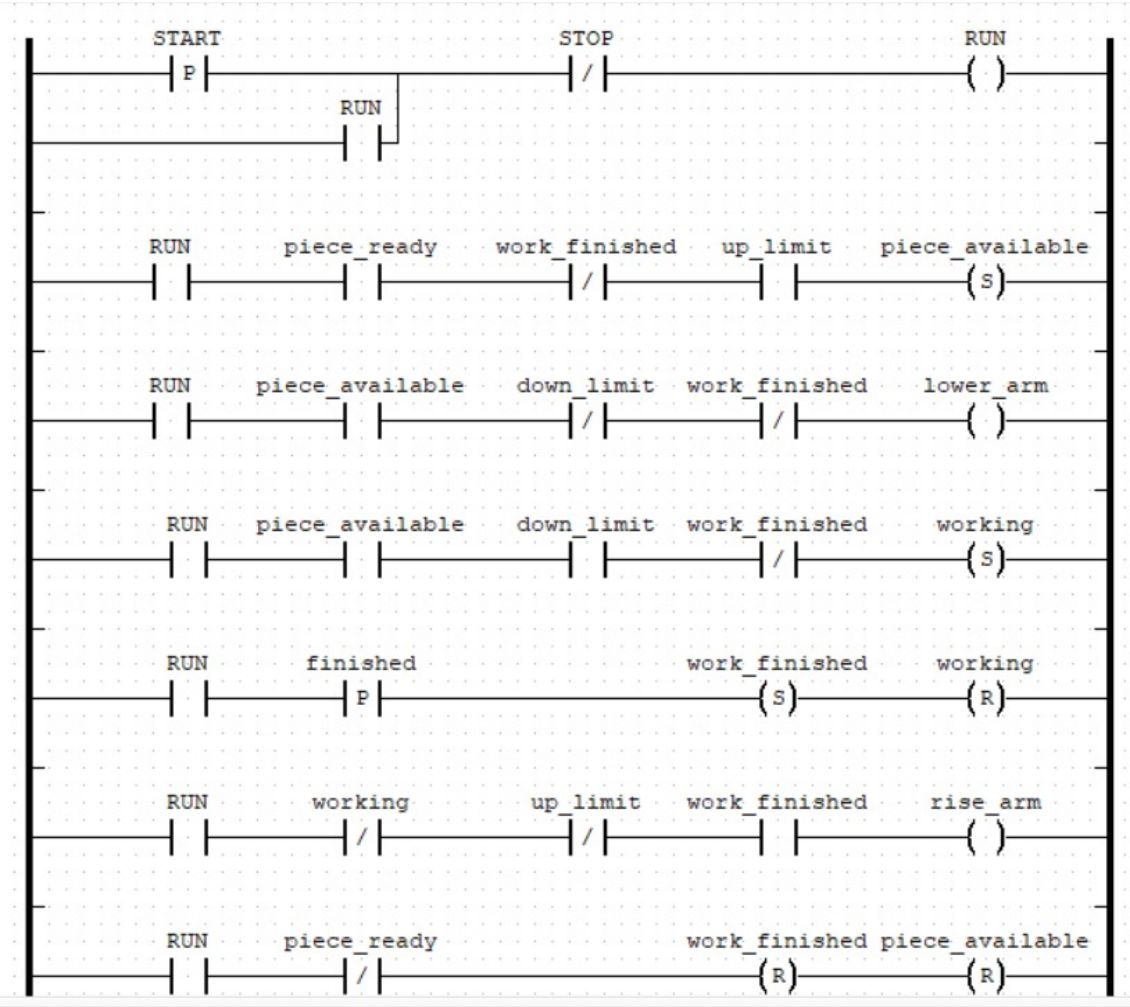

Figura 5.1: Programma Ladder Logic implementato nel PLC virtuale

Le variabili, mostrate in Figura 5.2, sono tutte di tipo booleano e rappresentano segnali di comando, variabili di stato e input del processo. Un aspetto rilevante riguarda la scelta di utilizzare esclusivamente variabili mappate come uscite (`%Q`) del PLC, anche per rappresentare segnali che, in un sistema reale, sarebbero tipicamente ingressi provenienti da sensori. Questa scelta è motivata dalla natura simulata del testbed: i segnali di campo non sono generati da hardware reale, ma da componenti software che emulano il comportamento fisico della linea industriale, che devono poter aggiornare dinamicamente i valori delle variabili associate al PLC tramite Modbus TCP. Per consentire questa interazione, le variabili sono state esposte in modo da risultare scrivibili dall'esterno, poiché le variabili mappate come ingressi (`%I`) non sono modificabili da componenti esterni, è stato necessario discostarsi dalla distinzione classica tra ingressi e uscite.

#	Nome	Classe	Yipo	Location	Initial Value	Opzione	Documentazione
1	START	Locale	BOOL	%QX0.0			
2	STOP	Locale	BOOL	%QX0.1			
3	RUN	Locale	BOOL	%QX0.2			
4	piece_ready	Locale	BOOL	%QX0.3			
5	work_finished	Locale	BOOL	%QX0.4			
6	up_limit	Locale	BOOL	%QX0.5	true		
7	piece_available	Locale	BOOL	%QX0.6			
8	down_limit	Locale	BOOL	%QX0.7			
9	lower_arm	Locale	BOOL	%QX0.8			
10	working	Locale	BOOL	%QX0.9			
11	finished	Locale	BOOL	%QX0.10			
12	rise_arm	Locale	BOOL	%QX0.11			

Figura 5.2: Mappatura delle variabili del PLC

Il funzionamento del programma può essere descritto come una sequenza di stati. In una fase iniziale, il PLC si trova in uno stato di attesa, nel quale il sistema non è in esecuzione. A seguito dell'attivazione del comando di avvio, il controllore entra nello stato operativo, mantenuto fino alla ricezione di un comando di arresto.
Quando un pezzo risulta disponibile, a seguito della segnalazione di un sensore di posizione virtuale, viene avviata la fase di lavorazione. Il PLC comanda quindi la discesa del braccio verso il pezzo e attiva una variabile interna che rappresenta lo stato di lavorazione in corso. Al completamento dell'operazione, segnalato da un evento opportuno, il sistema transita in uno stato di fine lavorazione. In questa condizione, viene comandato il movimento di risalita del braccio fino al raggiungimento della posizione iniziale. A questo punto, lo stato delle variabili interne viene resettato, dando avvio a un nuovo ciclo operativo.

Una volta compilato, l'editor genera una versione del programma in Structured Text (ST), utilizzato successivamente per la configurazione di OpenPLC come descritto nei paragrafi successivi.

Struttura del Dockerfile La realizzazione del container OpenPLC è basata su un Dockerfile personalizzato 5.1, progettato per automatizzare completamente il processo di installazione e configurazione del runtime e dell'interfaccia web. L'immagine viene costruita a partire da una base Linux minimale, sulla quale vengono installate le dipendenze necessarie e il codice sorgente di OpenPLC.

```
FROM debian:trixie-slim

WORKDIR /home/openplc

RUN apt update && apt install -y git sudo netcat-traditional python3-pip

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt --break-system-packages

RUN git clone https://github.com/thiagoralves/OpenPLC_v3.git
WORKDIR /home/openplc/OpenPLC_v3
RUN ./install.sh docker

COPY ./arm_plc.st /home/openplc/scripts/
COPY setup.sh /home/openplc
COPY start_plc.py /home/openplc
COPY run.sh /home/openplc

RUN sudo chmod +x /home/openplc/setup.sh
RUN /home/openplc/setup.sh

RUN sudo chmod +x /home/openplc/run.sh
CMD ["/home/openplc/run.sh"]
```

Codice 5.1: Dockerfile utilizzato per la costruzione del container OpenPLC

Durante la fase di build viene eseguito uno script di configurazione, al fine di ottenere un'istanza immediatamente operativa all'avvio del container, senza la necessità di interventi manuali. Questa scelta consente di garantire la completa riproducibilità dell'ambiente, assicurando che ogni istanza del PLC risulti identica e pronta all'esecuzione fin dal primo avvio.

Configurazione di OpenPLC Al fine di garantire un avvio completamente automatico del controllore, è stato sviluppato un insieme di script di configurazione eseguiti durante la fase di build e all'avvio del container.
In particolare, la sequenza di configurazione prevede:

- `setup.sh`: script shell responsabile del caricamento del programma in Structured Text (ST) all'interno del runtime OpenPLC. Lo script si occupa inoltre dell'inserimento delle informazioni necessarie nel database associato all'interfaccia web, mediante l'esecuzione di query SQL, al fine di rendere il programma visibile al server web.

- `start_plc.py` 5.2: script Python che interagisce con l'interfaccia web di OpenPLC per automatizzare le operazioni di compilazione e avvio del programma precedentemente caricato. In particolare, lo script simula le richieste normalmente effettuate tramite interfaccia utente, consentendo l'avvio del controllore senza interventi manuali.

```
import time
from dotenv import load_dotenv
import os

import requests
from bs4 import BeautifulSoup

load_dotenv()
s = requests.Session()

# login all'interfaccia con le credeziali di default di openplc e mantenimento
    della sessione
soup = BeautifulSoup(s.get("http://localhost:8080/login").text, "html.parser")
csrf = soup.find("input", {"name": "csrf_token"})["value"]
login = {"username": os.getenv("OPENPLC_ADMIIN"), "password": os.getenv(
    "OPENPLC_PASSWORD"), "csrf_token": csrf}

s.post("http://localhost:8080/login", data=login)

# compilazione del programma
```

```
s.get("http://localhost:8080/compile-program?file=arm_plc.st")

time.sleep(10) # attesa della compilazione
s.get("http://localhost:8080/start_plc") # avvio del plc
```

Codice 5.2: Configurazione automatizzata di OpenPLC

Hardware-in-the-Loop (HIL)

Nel testbed virtuale, il comportamento dei dispositivi di campo, quali bracci robotici, nastri trasportatori e sensori, è stato emulato mediante componenti software che implementano un modello di Hardware-in-the-Loop (HIL).
L'approccio HIL consente di integrare la logica di controllo reale, eseguita dai PLC, con un ambiente simulato in grado di riprodurre le dinamiche del sistema fisico.
I dispositivi simulati interagiscono con i PLC tramite protocollo Modbus TCP, operando come client che leggono e scrivono i registri esposti dai controllori. In particolare, i segnali che in un sistema reale sarebbero generati da sensori vengono simulati aggiornando direttamente le variabili dei PLC, mentre i comandi di attuazione vengono letti e utilizzati per determinare il comportamento del modello simulato.
I dispositivi HIL sono implementati come script Python eseguiti all'interno di container dedicati. Ciascun processo simula contemporaneamente il comportamento di un braccio robotico, di un nastro trasportatore e di un sensore di posizione, gestendo lo stato interno dei dispositivi e aggiornando i segnali di input e output in funzione delle condizioni operative.

Struttura del Dockerfile I container Docker dei dispositivi HIL sono basati su un'immagine Python minimale; l'utilizzo di una base leggera consente di ridurre l'overhead e migliorare i tempi di avvio dei componenti.
Il Dockerfile gestisce l'installazione delle dipendenze necessarie tramite il file `requirements.txt` e la copia del codice applicativo all'interno del container. All'avvio dei container, viene eseguito lo script principale che implementa la logica dei dispositivi simulati.

Script di simulazione La logica dei dispositivi HIL è implementata mediante uno script Python che realizza un ciclo continuo di interazione con il PLC tramite protocollo Modbus TCP. Il codice modella il comportamento dei dispositivi attraverso una sequenza di stati logici, aggiornando i segnali di ingresso e uscita in funzione dei comandi ricevuti dal controllore.
Per prima cosa viene importato il modulo `pyModbusTCP`, che fornisce un'interfaccia Python per la comunicazione tramite protocollo Modbus TCP. La libreria consente di eseguire operazioni di lettura e scrittura sui registri del PLC in modo diretto, mediante primitive di alto livello per la gestione dei coil e dei registri Modbus.

```
from pyModbusTCP.client import ModbusClient
```

Lo script stabilisce inizialmente una connessione con il PLC e attende che il sistema sia disponibile. Poiché il componente OpenPLC richiede diversi passaggi di configurazione durante la fase di avvio, è possibile che i container dei dispositivi HIL vengano eseguiti prima che il PLC sia completamente operativo.

```
client = ModbusClient(PLC_ADDRESS, 502)

while not client.open():
    print("waiting for " + PLC_ADDRESS + "...")
    time.sleep(0.5)

while True:
    try:
        client.read_coils(2, 1)[0]
        break
    except Exception as e:
        print(e)
```

Una volta connesso correttamente al server PLC, viene inviato il segnale di `START` per abilitare lo stato di operatività del PLC.

```
def start_plc():
    client.write_single_coil(0, True)
```

```
    time.sleep(0.5)  # start input
    client.write_single_coil(0, False)
```

Il comportamento del sistema è implementato come un ciclo di controllo continuo, nel quale il processo legge periodicamente lo stato delle variabili del PLC e determina le azioni da eseguire. In particolare, il modello implementa una logica a stati che simula le principali fasi operative della linea.

Le principali operazioni sono modellate mediante funzioni che rappresentano il comportamento dei dispositivi fisici, integrando inoltre ritardi temporali espliciti per simulare la durata delle operazioni, quali il movimento del braccio e la fase di lavorazione.

```
def rise_arm():
    print("Rising arm...")
    client.write_single_coil(7, False)  # unset down limit
    time.sleep(2)
    client.write_single_coil(5, True)  # set up limit
    print("Up limit reached")


def lower_arm():
    print("Lowering arm...")
    client.write_single_coil(5, False)  # unset up limit
    time.sleep(2)
    client.write_single_coil(7, True)  # set down limit
    print("Down limit reached")


def work():
    print("Working...")
    time.sleep(5)
    client.write_single_coil(10, True)
    time.sleep(0.5)  # finish input
    client.write_single_coil(10, False)
    print("Work finished")
```

Il sistema legge i comandi dal PLC, ciascuno dei quali è associato a una funzione. Allo stesso tempo, lo script scrive nei registri del PLC i segnali che rappresentano lo stato del sistema fisico, come i fine corsa e la presenza del pezzo. L'utilizzo di un ciclo di polling con intervallo temporale controllato consente di simulare un comportamento reattivo, mantenendo un equilibrio tra realismo e semplicità implementativa.

```
while client.open():
    run = client.read_coils(2, 1)[0]
    if run:
        rise_arm_signal = client.read_coils(11, 1)[0]
        up_limit_signal = client.read_coils(5, 1)[0]
        if rise_arm_signal and not up_limit_signal:
            rise_arm()

        working_signal = client.read_coils(9, 1)[0]
        if working_signal:
            work()

        lower_arm_signal = client.read_coils(8, 1)[0]
        down_limit_signal = client.read_coils(7, 1)[0]
        if lower_arm_signal and not down_limit_signal:
            lower_arm()

        work_finished_signal = client.read_coils(4, 1)[0]
        piece_available_signal = client.read_coils(6, 1)[0]

        if not work_finished_signal \
                and up_limit_signal \
                and not piece_available_signal:
            print("Waiting for piece...")
            time.sleep(2)
            client.write_single_coil(3, True)  # set piece ready
            print("Piece arrived")

        if work_finished_signal and up_limit_signal and piece_available_signal
```

```
            :
                client.write_single_coil(3, False)  # unset piece ready
                print("Piece gone")

    time.sleep(0.5)
```

Questo approccio consente di riprodurre in modo realistico l'interazione tra controllori e dispositivi di campo, mantenendo al contempo un elevato grado di flessibilità per l'esecuzione di scenari di test e validazione in ambiente controllato.

Sistema SCADA

Nel testbed virtuale, il sistema SCADA è implementato all'interno di un container mediante la piattaforma ScadaBR, che fornisce un'interfaccia web per la supervisione dell'intero processo industriale attraverso dashboard dedicate, interagendo con i PLC tramite protocollo Modbus TCP.

Configurazione dell'interfaccia ScadaBR La configurazione del sistema avviene mediante la definizione di *data source* 5.3 e *data point* 5.4. I data source rappresentano i dispositivi con cui il sistema SCADA comunica, nel caso specifico i PLC. I data point rappresentano invece le singole variabili di processo, associate agli indirizzi dei registri Modbus esposti dai controllori.

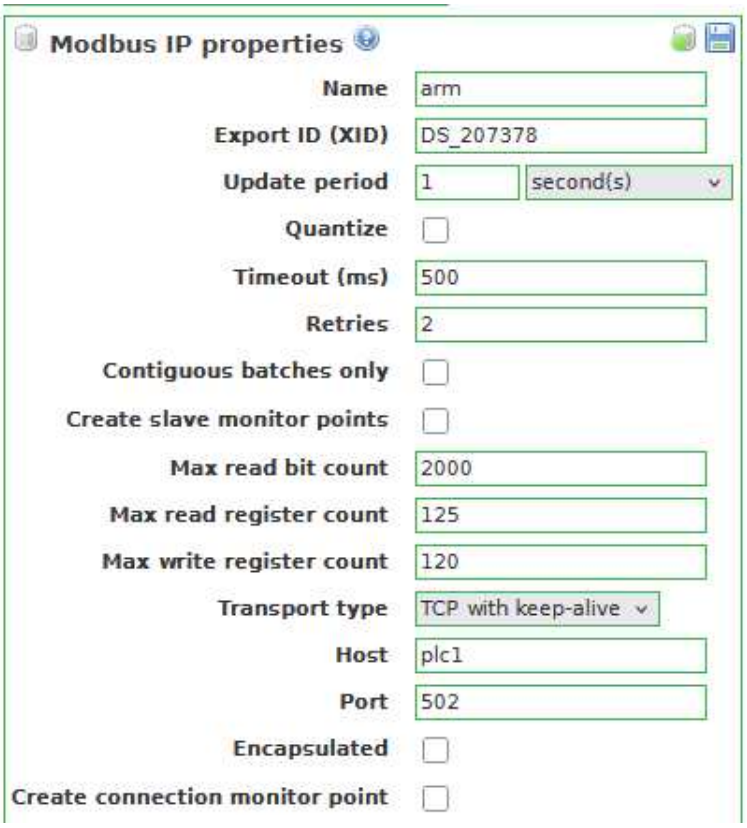

Figura 5.3: Configurazione di un data source Modbus in ScadaBR

Points

Name	Data type	Status	Slave	Range	Offset (0-based)	
down_limit	Binary		1	Coil status	7	
lower_arm	Binary		1	Coil status	8	
piece_available	Binary		1	Coil status	6	
rise_arm	Binary		1	Coil status	11	
run	Binary		1	Coil status	2	
stop	Binary		1	Coil status	1	
up_limit	Binary		1	Coil status	5	
work_finished	Binary		1	Coil status	4	
working	Binary		1	Coil status	9	

Figura 5.4: Configurazione dei data point associati a un PLC

Attraverso l'interfaccia è quindi possibile mappare direttamente le variabili dei PLC ai data point, consentendo la visualizzazione in tempo reale dello stato del processo e, ove previsto, l'invio di comandi verso i dispositivi di controllo.
Oltre alla configurazione dei dati, ScadaBR permette la creazione di dashboard di supervisione 5.5, utilizzate per rappresentare graficamente lo stato della linea e facilitare l'interazione con il sistema.

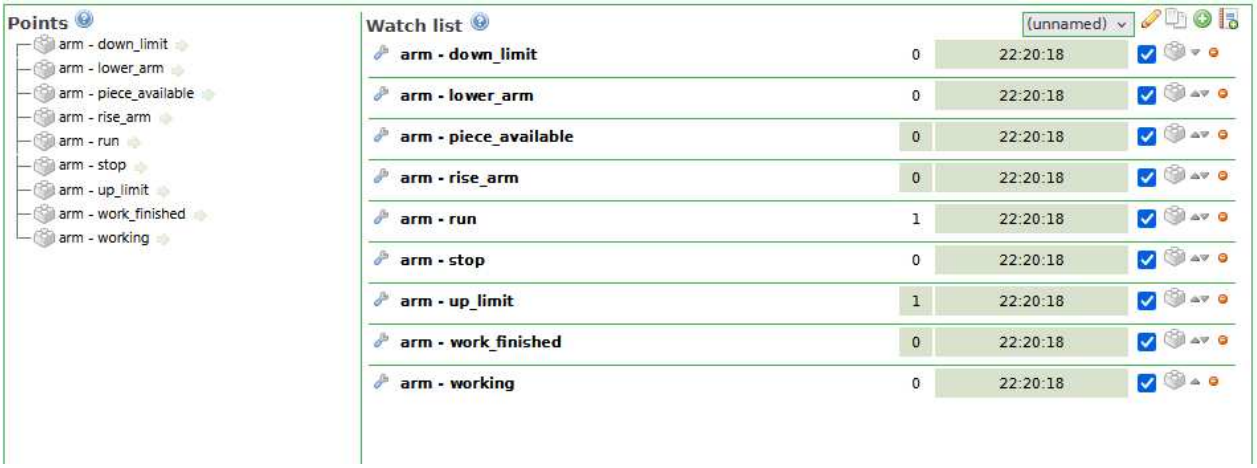

Figura 5.5: Dashboard di supervisione della linea

Nel sistema proposto, tali configurazioni sono predefinite e importate automaticamente all'interno del database di ScadaBR, evitando la necessità di una configurazione manuale tramite interfaccia web.

Struttura del Dockerfile L'immagine è costruita a partire da una base Debian minimale. Durante la fase di build viene scaricato il pacchetto di installazione di ScadaBR dalla release ufficiale del progetto e installato automaticamente all'interno del container.

Un aspetto rilevante della costruzione del container riguarda la predisposizione del database utilizzato da ScadaBR.
Apache Derby è un database relazionale embedded scritto in Java, utilizzato per la gestione dei dati e della configurazione del sistema. Essendo embedded, viene eseguito all'interno della stessa applicazione, senza richiedere un servizio esterno dedicato.
L'esecuzione del servizio è delegata allo script `entrypoint.sh`, che avvia il web server ScadaBR e, se il database non è ancora stato inizializzato, esegue lo script `db_setup.sh`. Quest'ultimo invoca

il comando `ij` sul file `import.del`, creando un file marcatore rendendo l'operazione idempotente, al fine di evitare importazioni ridondanti.

```
if [ ! -f "/opt/ScadaBR/tomcat/webapps/ScadaBR/db/scadabrDB/.import_done" ]; then
    /opt/ScadaBR/tomcat/bin/shutdown.sh
    while pgrep -f org.apache.catalina.startup.Bootstrap >/dev/null; do
        echo "waiting server to be down..."
        sleep 2
    done

    ij "/backup/import.del"
    touch "/opt/ScadaBR/tomcat/webapps/ScadaBR/db/scadabrDB/.import_done"
    /opt/ScadaBR/tomcat/bin/startup.sh
fi
echo "import done!"
```

Il comando `ij` è uno strumento fornito da Apache Derby per l'esecuzione di script SQL. Nel caso specifico, esso viene utilizzato per importare all'interno del database le configurazioni necessarie al funzionamento iniziale di ScadaBR. In questo modo, il container non viene avviato con un'istanza vuota, ma con una configurazione già predisposta, comprensiva degli elementi necessari alla supervisione del sistema.

```
CONNECT 'jdbc:derby:/opt/ScadaBR/tomcat/webapps/ScadaBR/db/scadabrDB';

CALL SYSCS_UTIL.SYSCS_IMPORT_TABLE('APP','DATASOURCES','/backup/datasources
    .del',null,null,null,0);
CALL SYSCS_UTIL.SYSCS_IMPORT_TABLE('APP','DATAPOINTS','/backup/datapoints.
    del',null,null,null,0);
CALL SYSCS_UTIL.SYSCS_IMPORT_TABLE('APP','WATCHLISTS','/backup/watchlists.
    del',null,null,null,0);
CALL SYSCS_UTIL.SYSCS_IMPORT_TABLE('APP','WATCHLISTPOINTS','/backup/
    watchlistpoints.del',null,null,null,0);

SHOW TABLES;
```

Codice 5.3: import.del

In particolare, vengono popolati i data source, i data point e le strutture di visualizzazione utilizzando la procedura `SYSCS_IMPORT_TABLE`, che consente il caricamento diretto dei dati da file precedentemente esportati dal database tramite `ij` e contenuti nella cartella `backup`.
Questo approccio consente di ottenere un'istanza SCADA immediatamente operativa fin dal primo avvio, riducendo al minimo le operazioni di configurazione manuale e garantendo la riproducibilità dell'ambiente.

Orchestrazione dell'infrastruttura

L'intero testbed virtuale è orchestrato mediante Docker Compose, attraverso un file di configurazione che definisce i servizi, le reti e i volumi necessari al funzionamento del sistema.
La comunicazione tra i componenti è realizzata attraverso una rete Docker dedicata, configurata con indirizzamento IP statico. Tale scelta consente di simulare una topologia di rete industriale realistica, nella quale ogni dispositivo è identificato da un indirizzo noto e stabile. Il driver `bridge` rappresenta la configurazione standard per reti Docker locali e consente la comunicazione diretta tra i container appartenenti alla stessa rete virtuale, isolandoli al contempo dal resto del sistema.

```
networks:
  ics:
    driver: bridge
    ipam:
      config:
        - subnet: 192.168.0.0/16
```

Ogni PLC espone l'interfaccia web su una porta differente del sistema host, consentendo l'accesso diretto alle HMI.

```
  plc1:
    pull_policy: never
    container_name: plc1
    build:
      context: ./plc
      dockerfile: Dockerfile
    env_file:
      - .env
```

```
    healthcheck:
      test: ["CMD", "test", "-f", "/tmp/.plc_ready"]
      interval: 2s
      timeout: 1s
      retries: 30
      start_period: 5s
    networks:
      ics:
        ipv4_address: 192.168.1.1
    ports:
      - 8081:8080
```

I dispositivi HIL dipendono esplicitamente dallo stato dei rispettivi PLC, sfruttando il meccanismo di `healthcheck`. In questo modo, l'avvio dei dispositivi simulati avviene solo quando il controllore associato risulta completamente operativo, evitando overhead dovuti al polling verso i PLC durante la fase di inizializzazione.

```
hil1:
  pull_policy: never
  container_name: hil1
  build:
    context: ./hil
    dockerfile: Dockerfile
  environment:
    - plc=1
  networks:
    ics:
      ipv4_address: 192.168.1.2
  depends_on:
    plc1:
      condition: service_healthy
```

Il sistema SCADA è anch'esso configurato per avviarsi solo dopo che tutti i PLC risultano disponibili. Il database utilizzato da ScadaBR è persistito tramite un volume Docker, garantendo la conservazione dello stato anche in caso di riavvio dei container. L'interfaccia web è esposta sulla porta 8080 dell'host.

```
  scada:
    pull_policy: never
    container_name: scada
    build:
      context: ./scada
      dockerfile: Dockerfile
    volumes:
      - scada:/opt/ScadaBR/tomcat/webapps/ScadaBR/db
    depends_on:
      plc1:
        condition: service_healthy
      plc2:
        condition: service_healthy
      plc3:
        condition: service_healthy
      plc4:
        condition: service_healthy
    networks:
      - ics
    ports:
      - 8080:8080

volumes:
  scada:
```

L'utilizzo di Docker Compose consente di avviare l'intera infrastruttura con un singolo comando, garantendo al contempo la riproducibilità dell'ambiente e la possibilità di riconfigurare rapidamente i diversi scenari sperimentali.

5.3.2 Integrazione di OpenZiti e WebAuthn

A partire dall'infrastruttura descritta nella sezione precedente, sono stati integrati un overlay network OpenZiti e un sistema di autenticazione passwordless tramite WebAuthn.
L'infrastruttura è stata estesa mediante l'introduzione di componenti aggiuntivi, anch'essi eseguiti all'interno di container Docker, che si integrano con i servizi esistenti senza modificarne il

comportamento interno.
La configurazione dell'infrastruttura è centralizzata tramite un file `.env`, utilizzato da Docker Compose per parametrizzare i diversi servizi del sistema. In particolare, tale file raccoglie le principali variabili di configurazione relative all'orchestrazione, ai componenti OpenZiti, ai servizi di autenticazione e ai parametri applicativi dei diversi container.
Questo approccio consente di separare la logica di deployment dai valori di configurazione, semplificando la gestione dell'ambiente e rendendo più agevole la riconfigurazione del sistema in scenari differenti.

In questa sezione verranno descritti nel dettaglio gli elementi introdotti e le modalità con cui essi interagiscono con l'infrastruttura OT, evidenziando il ruolo di OpenZiti nella gestione delle comunicazioni e di WebAuthn nel processo di autenticazione.

Componenti OpenZiti

Per la costruzione dei container dei componenti dell'overlay sono state utilizzate direttamente le immagini ufficiali messe a disposizione dal progetto OpenZiti. La loro configurazione e il loro avvio sono gestiti tramite il file `docker-compose.yaml`, senza la necessità di definire Dockerfile personalizzati.

Controller Il controller è stato deployato utilizzando l'immagine `openziti/quickstart`, che fornisce una configurazione predefinita dell'infrastruttura OpenZiti.

```
ziti-controller:
  image: ${ZITI_IMAGE}:${ZITI_VERSION}
  container_name: ziti-controller
  volumes:
    - ziti-ctrl:/persistent
  environment:
    - ZITI_CTRL_NAME=${ZITI_CTRL_NAME:-ziti-edge-controller}
    - ZITI_CTRL_EDGE_ADVERTISED_ADDRESS=${ZITI_CTRL_EDGE_ADVERTISED_ADDRESS:-ziti-
  edge-controller}
    - ZITI_CTRL_EDGE_ADVERTISED_PORT=${ZITI_CTRL_EDGE_ADVERTISED_PORT:-1280}
    - ZITI_CTRL_EDGE_IP_OVERRIDE=${ZITI_CTRL_EDGE_IP_OVERRIDE:-127.0.0.1}
```

```
    - ZITI_CTRL_ADVERTISED_PORT=${ZITI_CTRL_ADVERTISED_PORT:-6262}
    - ZITI_EDGE_IDENTITY_ENROLLMENT_DURATION=${
  ZITI_EDGE_IDENTITY_ENROLLMENT_DURATION}
    - ZITI_ROUTER_ENROLLMENT_DURATION=${ZITI_ROUTER_ENROLLMENT_DURATION}
    - ZITI_USER=${ZITI_USER:-admin}
    - ZITI_PWD=${ZITI_PWD}
  env_file:
    - ./.env
  ports:
    - target: ${ZITI_CTRL_ADVERTISED_PORT:-6262}
      published: ${ZITI_CTRL_ADVERTISED_PORT:-6262}
      host_ip: ${ZITI_INTERFACE:-0.0.0.0}
    - target: ${ZITI_CTRL_EDGE_ADVERTISED_PORT:-1280}
      published: ${ZITI_CTRL_EDGE_ADVERTISED_PORT:-1280}
      host_ip: ${ZITI_INTERFACE:-0.0.0.0}
  networks:
    ziti:
      aliases:
        - ziti-edge-controller
  entrypoint:
    - /var/openziti/scripts/run-controller.sh
  healthcheck:
    test: curl -m 1 -s -k -f https://${ZITI_CTRL_EDGE_ADVERTISED_ADDRESS:-ziti-edge-
  controller}:${ZITI_CTRL_EDGE_ADVERTISED_PORT:-1280}/edge/client/v1/version
    interval: 1s
    timeout: 3s
    retries: 30
```

La configurazione del servizio è gestita principalmente attraverso variabili d'ambiente, che consentono di parametrizzare dinamicamente il comportamento del controller senza modificare direttamente l'immagine del servizio.

In particolare, `ZITI_CTRL_NAME` definisce il nome logico del controller. Le variabili `ZITI\_CTRL\_EDGE\_ADVERTISED\_ADDRESS` e `ZITI\_CTRL\_EDGE\_ADVERTISED\_PORT` specificano l'indirizzo e la porta del piano edge, utilizzati dai componenti della rete overlay per raggiungere il controller. La variabile `ZITI_CTRL_ADVERTISED_PORT` definisce la porta del piano di controllo,

ovvero l'interfaccia a cui un amministratore può collegarsi tramite *ziti cli* per configurare il controller. Queste due variabili corrispondono alle uniche interfacce esposte dal controller sulla rete.
La variabile `ZITI_CTRL_EDGE_IP_OVERRIDE` viene utilizzata per aggiungere un indirizzo IP valido ai *Subject Alternative Name* del certificato edge del controller. Tale variabile deve essere definita correttamente fin dal primo avvio, poiché la generazione della PKI e dei certificati associati avviene una sola volta e un'impostazione errata comprometterebbe la validità dei certificati.
Le variabili `ZITI\_EDGE\_IDENTITY\_ENROLLMENT\_DURATION` e `ZITI\_ROUTER\_ENROLLMENT\_DURATION` definiscono invece la durata della finestra temporale, espressa in minuti, entro cui identità e router possono completare la procedura di enrollment. Nel sistema proposto entrambe sono impostate a `10080`, corrispondenti a sette giorni, in linea con i valori indicati dalla documentazione.
Infine, `ZITI_USER` e `ZITI_PWD` definiscono le credenziali amministrative iniziali del controller.

I dati del controller vengono persistiti tramite un volume Docker dedicato, garantendo la conservazione dello stato dell'infrastruttura, incluse identità, configurazioni e politiche di accesso, anche in caso di riavvio del servizio.
Il meccanismo di `healthcheck` consente di sincronizzare correttamente le fasi di avvio, garantendo che i componenti dell'overlay vengano avviati solo quando il controller risulta pienamente operativo.

Router Nella fase di progettazione dell'architettura era prevista la presenza di un componente dedicato al ruolo di edge router, responsabile dell'instradamento del traffico all'interno dell'overlay OpenZiti.
Nell'implementazione del testbed virtuale, tuttavia, tale componente non è stato deployato come servizio separato. I tunneler associati ai diversi componenti dell'infrastruttura OT, come descritto nei paragrafi successivi, sono stati infatti configurati per operare anche come edge router, sfruttando la flessibilità dell'ambiente simulato.
In questo modo, ciascun nodo della rete è in grado non solo di stabilire connessioni verso l'overlay, ma anche di partecipare al forwarding del traffico contribuendo alla formazione della mesh network, svolgendo simultaneamente il ruolo di client e di router.

Questa scelta consente di semplificare l'architettura complessiva, riducendo il numero di componenti da gestire, senza compromettere il corretto funzionamento della rete overlay nel contesto del testbed virtuale.

Tunneler L'integrazione dei componenti OT nell'overlay OpenZiti è realizzata mediante tunneler dedicati, implementati tramite container basati sull'immagine ufficiale `openziti/ziti-router`. Ciascun tunneler è affiancato a uno specifico componente dell'infrastruttura OT attraverso l'adozione del pattern *sidecar* e viene avviato solo dopo il completamento con successo del relativo container di inizializzazione, a sua volta eseguito dopo l'avvio del controller.
In Docker Compose, la configurazione *sidecar* è implementata mediante la direttiva `network\_mode:service:<nome-servizio>`, che consente a due container di condividere lo stesso namespace di rete. In questo modo, il servizio applicativo e il relativo tunneler operano come se fossero eseguiti all'interno dello stesso nodo di rete.

```
hil1:
  build:
    context: ./hil
    dockerfile: Dockerfile
  container_name: hil1
  depends_on:
    plc1:
      condition: service_healthy
    ziti-tunneler-hil1:
      condition: service_healthy
  environment:
  - plc=1
  network_mode: service:ziti-tunneler-hil1
```

Il tunneler diventa quindi l'unico punto di accesso alla rete per il componente, intercettando tutto il traffico in ingresso e in uscita in modo trasparente rispetto all'applicazione.
Un ulteriore aspetto rilevante riguarda il meccanismo di risoluzione dei servizi all'interno dell'overlay OpenZiti. Il tunneler sidecar consente infatti di accedere ai servizi esposti tramite gli hostname definiti nelle configurazioni `intercept.v1`.

Ciò implica che i servizi non vengano più raggiunti tramite indirizzi IP o nomi dei container Docker, ma mediante nomi logici propri dell'overlay. Di conseguenza, i componenti dell'infrastruttura devono essere configurati per utilizzare tali hostname, demandando al tunneler la risoluzione e l'instradamento del traffico verso il servizio corretto. Per questo motivo, nel file `docker-compose.yaml` non è più necessario assegnare indirizzi IP statici ai servizi.

Ogni istanza utilizza un volume Docker dedicato per la persistenza della configurazione locale e dei certificati.
Lo script `enroll.sh`, montato all'interno del container, sovrascrive l'entrypoint per gestire la fase di enrollment prima dell'avvio del componente.

```
...
while [ ! -f "/ziti-router/enroll.jwt" ] && [ ! -f "/ziti-router/config.yml" ]; do
  echo "waiting jwt..."
  sleep 1
done

if [ ! -f "/ziti-router/config.yml" ]; then
  echo "enrollment..."
  ZITI_ENROLL_TOKEN="$(cat /ziti-router/enroll.jwt)"
  export ZITI_ENROLL_TOKEN
else
  unset ZITI_ENROLL_TOKEN
fi
...
```

Codice 5.4: script enroll.sh

La fase di enrollment produce il file di configurazione `config.yml`. Se tale file non è presente, lo script attende la disponibilità del token JWT generato dal container di inizializzazione e lo esporta come variabile d'ambiente, consentendo all'immagine ufficiale OpenZiti di completare automaticamente la procedura di enrollment.
Il meccanismo di `healthcheck`, basato sul comando `ziti agent stats`, consente di verificare la corretta operatività del tunneler dopo l'avvio.

Un aspetto rilevante dell'implementazione riguarda la differenziazione delle modalità operative dei tunneler in funzione del componente. I tunneler collegati ai dispositivi HIL e al sistema SCADA sono configurati in modalità `tproxy`, mentre quelli associati ai PLC operano in modalità `host`.
La modalità `tproxy` è utilizzata nei casi in cui il tunneler deve intercettare e gestire il traffico di rete del servizio, operando in modo trasparente rispetto all'applicazione. Per questo motivo, tali container vengono eseguiti con privilegi aggiuntivi, in particolare `NET_ADMIN`, e con privilegi di amministrazione, necessari alla manipolazione dello stack di rete del container stesso.
La modalità `host` viene invece adottata per i tunneler associati ai PLC, nei quali il componente OpenZiti è configurato per esporre il servizio verso l'overlay, rendendo raggiungibili le interfacce applicative e i servizi Modbus dei controllori senza richiedere modifiche al loro funzionamento interno.

Container di inizializzazione Un aspetto ricorrente dell'implementazione riguarda l'utilizzo di coppie di container, costituite da un componente di inizializzazione e dal corrispondente componente operativo. L'inizializzatore viene eseguito in una fase preliminare e ha il compito di predisporre i file, le credenziali e le configurazioni necessarie all'esecuzione del componente principale.

```
ziti-controller-init-container:
  image: ${ZITI_IMAGE}:${ZITI_VERSION}
  depends_on:
    ziti-controller:
      condition: service_healthy
  volumes:
    - ziti-ctrl:/persistent
  environment:
    - ZITI_CTRL_EDGE_ADVERTISED_ADDRESS=${ZITI_CTRL_EDGE_ADVERTISED_ADDRESS:-ziti-
  edge-controller}
    - ZITI_CTRL_EDGE_ADVERTISED_PORT=${ZITI_CTRL_EDGE_ADVERTISED_PORT:-1280}
  env_file:
    - ./.env
  networks:
    ziti: null
  command:
    - /var/openziti/scripts/access-control.sh
```

```
    entrypoint:
      - /var/openziti/scripts/run-with-ziti-cli.sh
```

Questo pattern è applicato anche ai tunneler, per i quali i container di inizializzazione svolgono un ruolo fondamentale nel processo di enrollment all'interno dell'overlay.
In questo caso, l'immagine utilizzata è `openziti/ziti-cli`, che consente al container di interagire con il controller tramite la CLI di OpenZiti. L'*entrypoint* viene sovrascritto con lo script personalizzato `bootstrap.sh`.

```
  ziti-tunneler-hil-init1:
    image: openziti/ziti-cli:1.6.12
    container_name: ziti-tunneler-hil-init1
    depends_on:
      ziti-controller:
        condition: service_healthy
    volumes:
      - ziti-client1:/ziti-router
      - ./ziti/bootstrap_tunneler_client.sh:/bootstrap.sh
    environment:
      - ZITI_USER=${ZITI_USER:-admin}
      - ZITI_PWD=${ZITI_PWD}
      - ZITI_CTRL_EDGE_ADVERTISED_PORT=${ZITI_CTRL_EDGE_ADVERTISED_PORT:-}
      - PLC=1
    networks:
      - ziti
    entrypoint:
      - /bin/sh
      - /bootstrap.sh
```

Lo script esegue innanzitutto l'autenticazione verso il controller OpenZiti, quindi verifica la presenza del token JWT all'interno del volume condiviso con il container principale. In assenza di tale file, vengono creati un nuovo edge router e la relativa identità, generando il token di enrollment (`enroll.jwt`).

```
#!/bin/sh
```

```
ziti edge login https://ziti-edge-controller:"${ZITI_CTRL_EDGE_ADVERTISED_PORT}" \
    --username="${ZITI_USER}" \
    --password="${ZITI_PWD}" \
    --yes

if [ ! -f /ziti-router/enroll.jwt ]; then
    echo 'Create HIL router'
    ziti edge create edge-router "hil-router${PLC}" \
        --tunneler-enabled \
        --jwt-output-file /ziti-router/enroll.jwt
    ziti edge update identity "hil-router${PLC}" \
        --role-attributes hil"${PLC}"

    ...
fi
...
```

Codice 5.5: creazione router e identità dei tunneler HIL in bootstrap_tunneler_client.sh

Nel caso dei tunneler in modalità `host` associati ai servizi esposti, come i PLC, gli script definiscono anche configurazioni di tipo `host.v1` e `intercept.v1`, un servizio OpenZiti e le relative *service policy*. In questo modo, il servizio viene pubblicato nell'overlay con un hostname logico dedicato, rendendolo accessibile agli altri componenti autorizzati.

```
...
    ziti edge create config "plc-config${PLC}" host.v1 \
        '{"protocol":"tcp", "address":"127.0.0.1","port":502}'
    ziti edge create config "hil-config${PLC}" intercept.v1 \
        "{\"protocols\":[\"tcp\"],\"addresses\":[\"ziti.plc${PLC}\"],
    \"portRanges\":[{\"low\":502, \"high\":502}]}"

    ziti edge create service "plc-service${PLC}" \
        --configs hil-config"${PLC}",plc-config"${PLC}" \
        --role-attributes plc-services"${PLC}"

    ziti edge create service-policy "plc-policy${PLC}" Bind \
        --service-roles "#plc-services${PLC}" \
```

```
        --identity-roles "#plc${PLC}"
...
```

Codice 5.6: creazione configurazioni host e intercept, servizio e service policy dei tunneler PLC in bootstrap_tunneler_plc.sh

Nel caso dei tunneler associati ai componenti client, come i dispositivi HIL, gli script definiscono invece le identità, gli attributi di ruolo e le policy di tipo `Dial`, che determinano i servizi raggiungibili all'interno dell'overlay.

```
...
    ziti edge create service-policy "hil-policy${PLC}" Dial \
        --service-roles "#plc-services${PLC}" \
        --identity-roles "#hil${PLC}"
...
```

Codice 5.7: creazione service policy dei tunneler HIL in bootstrap_tunneler_client.sh

Tale approccio consente di separare la configurazione iniziale dal normale ciclo di esecuzione dei servizi, migliorando la modularità del deployment e la riproducibilità dell'infrastruttura.

Componenti WebAuthn

L'integrazione è stata realizzata introducendo componenti applicativi dedicati, anch'essi containerizzati, responsabili della gestione delle fasi di registrazione e autenticazione degli utenti. Tali componenti operano in sinergia con il reverse proxy NGINX, che agisce come punto di controllo per l'accesso ai servizi protetti.

Database Il sistema di autenticazione si basa su un database PostgreSQL dedicato, implementato tramite un container basato sull'immagine ufficiale `postgres`, che fornisce un'istanza PostgreSQL preconfigurata e pronta all'uso.

```
db_user_handler:
  image: postgres:18.1
  container_name: db_user_handler
  depends_on:
```

```
      ziti-tunneler-db:
        condition: service_healthy
    volumes:
      - db_user_handler:/var/lib/postgresql
    environment:
      POSTGRES_USER: ${DATABASE_USER}
      POSTGRES_PASSWORD: ${DATABASE_PASSWORD}
      POSTGRES_DB: app
    network_mode: service:ziti-tunneler-db
```

La configurazione del servizio è realizzata principalmente tramite variabili d'ambiente. In particolare, `POSTGRES_USER`, `POSTGRES_PASSWORD` e `POSTGRES_DB` definiscono rispettivamente le credenziali di accesso e il database iniziale creato all'avvio del container.
I dati del database sono persistiti attraverso un volume, garantendo la conservazione delle informazioni anche in caso di riavvio o ricreazione del container.

Anche questo componente è stato integrato nell'overlay OpenZiti mediante un tunneler configurato come *sidecar*. A tal fine, il container di inizializzazione associato al tunneler provvede alla definizione delle configurazioni necessarie alla pubblicazione del servizio.

```
...
    ziti edge create edge-router db-router \
        --tunneler-enabled \
        --jwt-output-file /ziti-router/enroll.jwt

    ziti edge update identity "db-router" \
        --role-attributes db
    ziti edge create config "db-config" host.v1 \
        '{"protocol":"tcp", "address":"127.0.0.1","port":5432}'
    ziti edge create config "userhandler-config" intercept.v1 \
        "{\"protocols\":[\"tcp\"],\"addresses\":[\"ziti.db\"],
    \"portRanges\":[{\"low\":5432, \"high\":5432}]}"

    ziti edge create service "db-service" \
        --configs userhandler-config,db-config \
        --role-attributes db-services
```

```
    ziti edge create service-policy "db-policy" Bind \
        --service-roles "#db-services" \
        --identity-roles "#db"
...
```

Codice 5.8: configurazione del servizio database tramite OpenZiti

La configurazione `host.v1` definisce il servizio locale esposto dal container, in questo caso il database in ascolto su `127.0.0.1:5432`. La configurazione `intercept.v1` associa invece al servizio un hostname logico (`ziti.db`), utilizzato dai componenti client per accedere al database attraverso l'overlay.
Il servizio OpenZiti così definito viene quindi reso disponibile ai componenti autorizzati tramite una *service policy* di tipo `Bind`, che consente al tunneler associato al database di esporre il servizio all'interno dell'overlay.

LoginGateway Il servizio LoginGateway è implementato come applicazione web basata su Flask, eseguita all'interno di un container costruito a partire dall'immagine `python:3.13`. L'accesso al database è gestito tramite SQLAlchemy, che consente di mappare le entità applicative su strutture relazionali e di astrarre le operazioni di persistenza. In particolare, il modello `User` è definito come classe e rappresenta la tabella principale utilizzata dal sistema di autenticazione.
Le operazioni di accesso ai dati, come il recupero dell'utente durante la fase di login, vengono effettuate mediante query ORM, evitando l'utilizzo diretto di query SQL. Questo approccio migliora la leggibilità del codice, favorisce la manutenibilità e consente una gestione più strutturata dello stato applicativo.

Lo scopo principale del LoginGateway è fornire una pagina di login tramite cui gli utenti possano autenticarsi senza l'utilizzo di password.

Figura 5.6: Pagina di login

Dopo l'inserimento del nome utente e l'avvio della procedura di autenticazione, il frontend invia una richiesta POST al backend, includendo lo username nel corpo della richiesta.

```
...
    const response = await fetch('/userlogin/autentication-options',
        {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ username: username })
        }
    );
...
```

Dopo aver verificato l'esistenza dell'utente nel database, il server genera le opzioni di autenticazione tramite la libreria Python webauthn, specificando l'identificativo del relying party. Tra tali opzioni è inclusa una challenge crittografica, che viene salvata nella sessione associata all'identificativo dell'utente. Le opzioni vengono quindi restituite al frontend.

```
@userlogin.route('/autentication-options', methods=['POST'])
def authenticate():
    username = request.json.get('username')
    user = User.query.filter(User.username == username).first()
    if not user:
        return jsonify({'error': "utente non trovato"}), 500
    options = webauthn.generate_authentication_options(
        rp_id=_hostname(),
        user_verification=UserVerificationRequirement.PREFERRED
```

```
    )
    session[f"challenge_{user.id}"] = options.challenge
    jsonOptions = webauthn.options_to_json(options)
    return jsonify(jsonOptions)
```

Ricevute le opzioni di autenticazione, il client invoca la funzione `startAuthentication` fornita dalla libreria *SimpleWebAuthn*.

```
authenticationOptions = await response.json();
jsonOptions = JSON.parse(authenticationOptions);

let asseResp;
try {
    asseResp = await startAuthentication(jsonOptions);
} catch (error) {
    alert("Qualcosa e' andato storto");
    console.error(error)
}
```

Tale funzione attiva il meccanismo di autenticazione WebAuthn nel browser, che richiede all'utente di confermare la propria identità tramite un autenticatore registrato, ad esempio un dispositivo mobile, una chiave hardware o un sensore biometrico. L'autenticatore esegue quindi la firma crittografica della challenge utilizzando la chiave privata associata all'utente e restituisce al client la credenziale risultante.

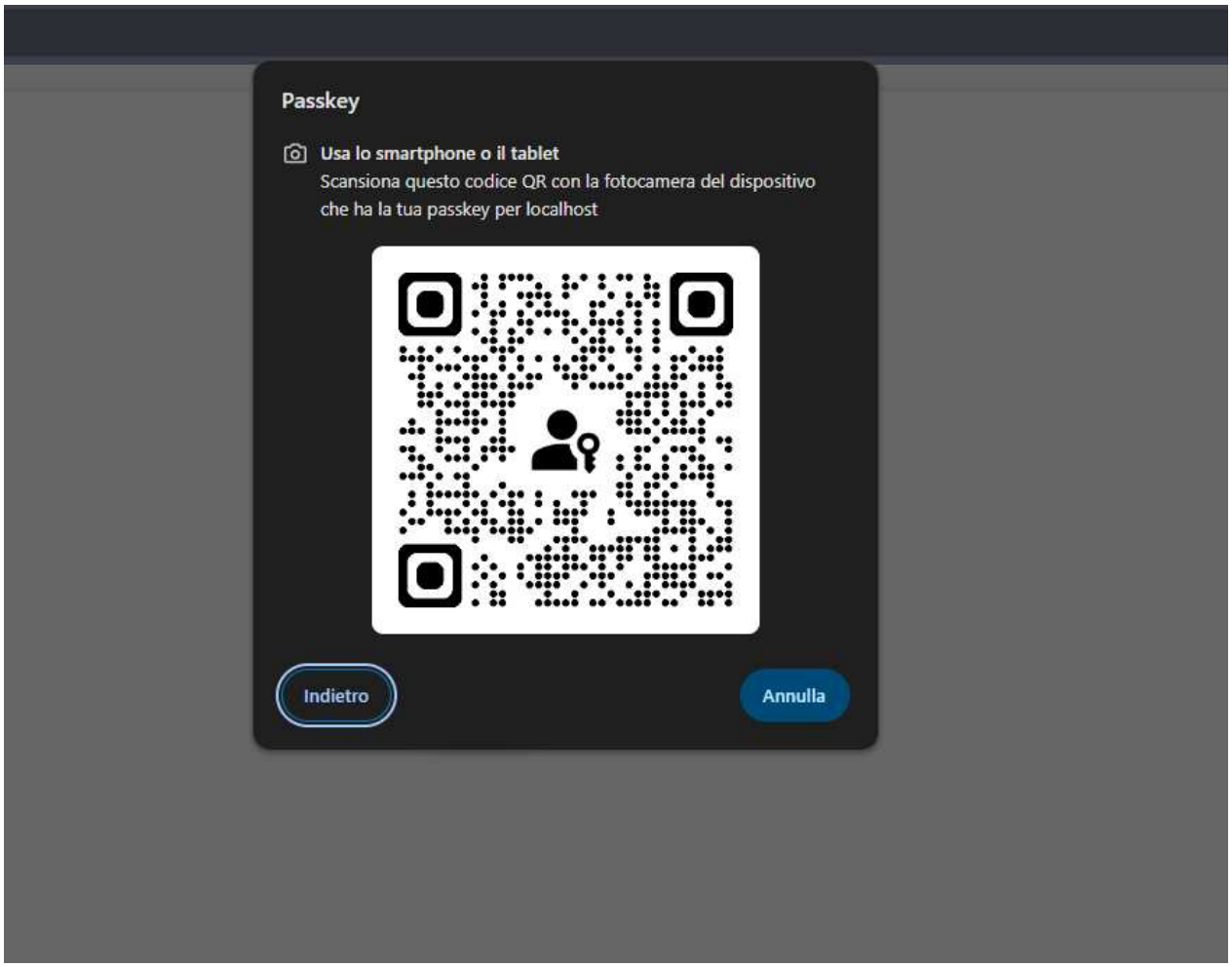

Figura 5.7: Codice QR per l'autenticazione

La credenziale prodotta dall'autenticatore viene quindi inviata nuovamente al backend per la verifica.

```
const authenticationResponse = await fetch('/userlogin/login-verify', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(
        {
            credential: asseResp,
            username: username
        }
    )
});
```

Nella fase finale del processo di autenticazione, il backend riceve la credenziale WebAuthn generata dal browser. Esso recupera l'utente associato allo username fornito e la challenge precedentemente salvata nella sessione.

Tramite la libreria webauthn, esegue quindi la verifica crittografica della credenziale ricevuta. In particolare, vengono controllati l'origine della richiesta, l'identificativo del relying party, la corrispondenza della challenge e la validità della firma, utilizzando la chiave pubblica associata all'utente e il contatore delle autenticazioni precedenti.

```
@userlogin.route('/login-verify', methods=['POST'])
def login_verify():
    try:
        data = request.json

        credential = data.get('credential')
        if not credential:
            return jsonify({'error': "credential non presente"}), 500

        username = data.get('username')
        user = User.query.filter(User.username == username).first()
        if not user:
            return jsonify({'error': "utente non trovato"}), 500

        challenge = session.get(f"challenge_{user.id}")
        if not challenge:
            return jsonify({'error': "challenge non presente"}), 500

        host = request.headers.get("Origin", request.host)
        webauthn.verify_authentication_response(
            credential=credential,
            expected_origin=f"{host}",
            expected_rp_id=_hostname(),
            expected_challenge=session.get(f"challenge_{user.id}"),
            credential_current_sign_count=user.current_sign_count,
            credential_public_key=user.credential_public_key,
            require_user_verification=True
```

```
        )

        login_user(user)
        return jsonify({'success': True, 'user': user.id}), 200
    except Exception as e:
        return jsonify(
            {'error': f'Errore durante la verifica delle credenziali. {e}'}
        ), 500
```

In caso di esito positivo, l'utente viene autenticato all'interno dell'applicazione tramite il meccanismo di sessione. In caso contrario, la richiesta viene rifiutata e il processo di autenticazione termina con un errore.

Analogamente agli altri componenti dell'infrastruttura, il servizio LoginGateway è integrato nell'overlay OpenZiti mediante un tunneler dedicato, implementato come container sidecar.
All'interno dell'overlay, il LoginGateway assume il ruolo di client, in quanto necessita di accedere al servizio del database.
Durante la fase di bootstrap del tunneler, viene quindi definita una *service policy* di tipo `Dial`, che autorizza l'identità associata al LoginGateway a stabilire connessioni verso il servizio del database.

```
ziti edge create service-policy "logingateway-policy" Dial \
    --service-roles "#db-services" \
    --identity-roles "#logingateway"
```

UserHandler Il servizio UserHandler è implementato come applicazione web basata su Flask, eseguita all'interno di un container costruito a partire dall'immagine `python:3.13`. Analogamente al LoginGateway, anche lo UserHandler interagisce con il database tramite SQLAlchemy, utilizzando modelli ORM per la gestione delle entità applicative.
In fase di primo avvio, lo UserHandler implementa un meccanismo di inizializzazione che richiede l'inserimento di un token segreto, configurato tramite file `.env`, per autorizzare la creazione dell'utente amministratore.

Figura 5.8: Primo accesso a UserHandler

Questo approccio elimina la necessità di credenziali predefinite e riduce il rischio di accessi non autorizzati nelle fasi iniziali del deployment, vincolando la creazione dell'account privilegiato al possesso del token.

La fase di autenticazione nello UserHandler segue lo stesso meccanismo descritto per il LoginGateway, basato sul protocollo WebAuthn. Pertanto, non viene qui ripetuta nel dettaglio.
Una volta autenticato, l'amministratore ha la possibilità di gestire gli account presenti nel sistema, in particolare creando nuovi utenti o rimuovendo quelli esistenti.

Figura 5.9: Pagina principale dello UserHandler

Il processo di registrazione segue una logica analoga a quella dell'autenticazione, ma con finalità diverse.

```
const REGISTER_USER_URL = "/usermanagement/register-user"
```

```
...
    try {
        const response = await fetch(REGISTER_USER_URL, {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(userData)
        });
...
```

A seguito della richiesta di registrazione, il backend aggiunge una nuova entità nel database e genera una serie di opzioni contenenti una challenge crittografica e le informazioni necessarie alla creazione di una nuova credenziale.

```
def _register_user(username, name, role=Role.USER.value):
    user = User(username=username, name=name,
                credential_public_key=b'', credential_id=b'', role=role)
    try:
        db.session.add(user)
        db.session.commit()
    except IntegrityError as e:
        db.session.rollback()
        if "duplicate key value" in str(e.orig):
            msg = "Username gia' esistente"
        else:
            msg = str(e.orig)
        return {'status': 'error', 'message': msg}, 400

    public_credential_creation_options =\
        webauthn.generate_registration_options(
            rp_id=_hostname(),
            rp_name="User Handler",
            user_name=user.username,
            authenticator_selection=AuthenticatorSelectionCriteria(
                resident_key=ResidentKeyRequirement.REQUIRED,
                user_verification=UserVerificationRequirement.PREFERRED)
        )
```

```
    session[f"challenge_{user.id}"] = \
        public_credential_creation_options.challenge
    return public_credential_creation_options, user.id
```

Le opzioni generate vengono inviate al client, che invoca la funzione `startRegistration` fornita dalla libreria *SimpleWebAuthn*. Tale funzione interagisce con l'autenticatore dell'utente per generare una nuova coppia di chiavi crittografiche.

```
...
const result = await response.json();
if (response.ok) {
    user_id = result.user_id;
    jsonOptions = JSON.parse(result.publicCredentialCreationOptions);

    let webauthn_response;
    webauthn_response = await SimpleWebAuthnBrowser.startRegistration(
       jsonOptions);
...
```

Durante questa fase, la chiave privata viene generata e mantenuta all'interno dell'autenticatore . Il backend riceve esclusivamente la chiave pubblica e le informazioni associate, che vengono memorizzate nel database.

```
...
const VERIFY_SAVE_CREDENTIAL_URL = "/usermanagement/verify-save-credentials"
...
    const verifyResponse = await fetch(VERIFY_SAVE_CREDENTIAL_URL, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
            credential: webauthn_response,
            challenge: Uint8Array.from(jsonOptions.challenge, c => c.
               charCodeAt(0)),
            user_id: user_id
        })
...
```

La risposta prodotta dal client viene quindi inviata al backend, che ne verifica la validità tramite la libreria `webauthn`. In caso di esito positivo, la chiave pubblica e i parametri associati vengono salvati nel database, completando la registrazione dell'utente.

```
def _verify_save_credentials(user_id, credential):
    result = webauthn.verify_registration_response(
        credential=credential,
        expected_challenge=session.get(f"challenge_{user_id}"),
        expected_origin=f"{request.scheme}://{request.host}",
        expected_rp_id=_hostname()
    )

    user = User.query.get(user_id)
    if not user:
        return {'status': 'error', 'message': 'Utente non trovato'}, 404

    user.credential_public_key = result.credential_public_key
    user.credential_id = result.credential_id
    user.current_sign_count = result.sign_count
    db.session.commit()
    return {'status': 'ok'}, 200
```

Anche il servizio UserHandler è integrato nell'overlay OpenZiti mediante un tunneler configurato come container sidecar. All'interno dell'overlay, esso assume un ruolo analogo a quello del LoginGateway, accedendo al database tramite una *service policy* di tipo `Dial`.

```
ziti edge create service-policy "userhandler-policy" Dial \
    --service-roles "#db-services" \
    --identity-roles "#userhandler"
```

NGINX Nel testbed virtuale, a differenza di quanto previsto nella fase di progettazione, l'accesso ai servizi protetti è centralizzato attraverso un unico server NGINX. Tale componente rappresenta il solo punto di ingresso verso le interfacce applicative esposte nell'infrastruttura, semplificando l'architettura complessiva e rendendo più immediata la gestione del controllo degli accessi.

Il reverse proxy è implementato tramite Docker Compose ed è basato sull'immagine ufficiale `nginx:stable-alpine3.23`, scelta che consente di utilizzare un web server minimale e leggero senza richiedere una build personalizzata del container.

```
nginx:
  image: nginx:stable-alpine3.23
  container_name: nginx
  depends_on:
    ziti-tunneler-nginx:
      condition: service_healthy
  volumes:
    - ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro
  network_mode: service:ziti-tunneler-nginx
```

Il server viene posto in ascolto sulla porta `5050` e instrada le richieste verso i servizi interni dell'infrastruttura, applicando preventivamente una verifica di autenticazione.
La configurazione del proxy è montata dall'esterno tramite volume in sola lettura. Il file definisce due location interne, `/_auth` e `/_login`, utilizzate rispettivamente per la verifica dell'autenticazione e per il reindirizzamento verso la pagina di login.

```
location /_auth {
    internal;
    proxy_pass http://ziti.logingateway:5000/auth;

    proxy_set_header Cookie $http_cookie;
    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
}

location /_login {
    internal;
    return 302 http://localhost:5050/login_?next=$request_uri;
}
```

La location `/_auth` non è raggiungibile direttamente dal client, in quanto marcata come `internal`, ed è utilizzata esclusivamente da NGINX per interrogare il LoginGateway tramite l'endpoint `/auth`. L'inoltro dei cookie e degli header di forwarding consente al servizio di autenticazione di verificare correttamente lo stato della sessione dell'utente.
La location `/_login` è anch'essa interna e viene utilizzata come target di reindirizzamento nel caso in cui la verifica di autenticazione fallisca. In particolare, l'utente viene rediretto verso la pagina di login includendo nella query string il parametro `next`, che consente di preservare la risorsa originariamente richiesta.

Le interfacce OT sono esposte tramite location dedicate, ad esempio `/plc1/`, `/plc2/`, `/plc3/`, `/plc4/` e `/ScadaBR/`. Ciascuna di esse applica il meccanismo `auth_request`, che delega a `/_auth` la verifica preventiva dell'autenticazione.

```
location /plc1/ {
    auth_request /_auth;
    error_page 401 = /_login;

    proxy_pass http://ziti.plc1.webserver:8080/;
    proxy_redirect / /plc1/;

    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;

    sub_filter_once off;
    sub_filter '/static' '/plc1/static';
}
```

In questo schema, ogni richiesta verso il servizio viene verificata dal LoginGateway prima di essere inoltrata al backend. Se il gateway restituisce un codice `401`, NGINX reindirizza il client verso la pagina di autenticazione; in caso contrario, la richiesta viene inoltrata al servizio OT corrispondente.

Il container NGINX condivide lo stack di rete con il relativo tunneler OpenZiti, operando interamente all'interno dell'overlay. Poiché il reverse proxy deve poter raggiungere molteplici servizi interni, vengono definite opportune *service policy* di tipo `Dial`, che autorizzano l'identità associata al tunneler di NGINX ad accedere ai servizi dell'overlay.

```
...
ziti edge create service-policy "webserver-client-policy${PLC}" Dial \
    --service-roles "#plc-webserver-services${PLC}" \
    --identity-roles "#nginx"
...
ziti edge create service-policy "scada-host-dial-policy" Dial \
    --service-roles "#scada-host-services" \
    --identity-roles "#nginx"
...
```

Dal punto di vista architetturale, il tunneler intercetta le richieste generate da NGINX e le instrada all'interno dell'overlay OpenZiti, garantendo che tutte le comunicazioni tra il reverse proxy e i servizi backend avvengano esclusivamente attraverso il canale sicuro.

Un aspetto rilevante riguarda la configurazione dei servizi LoginGateway e SCADA. Affinché il reverse proxy NGINX possa accedere alle relative interfacce web attraverso l'overlay OpenZiti, è stato necessario configurare un ulteriore tunneler in modalità `host`, con i relativi service policy, servizio e configurazioni.
In questo modo, tali servizi vengono esplicitamente pubblicati all'interno dell'overlay e risultano raggiungibili dal reverse proxy stesso.

5.4 Testbed Fisico

Il testbed fisico ha l'obiettivo di valutare il comportamento del sistema in un contesto più realistico, caratterizzato da risorse limitate, vincoli hardware e condizioni operative più vicine a quelle di un'infrastruttura OT reale.
A differenza della simulazione virtuale, in cui tutti i componenti sono eseguiti come container su un unico host, l'ambiente fisico prevede la distribuzione dei servizi su dispositivi distinti, nello

specifico Raspberry Pi. Questa configurazione consente di riprodurre una topologia distribuita, in cui i diversi componenti comunicano attraverso la rete e non condividono lo stesso ambiente di esecuzione.
L'architettura logica del sistema rimane invariata: i servizi continuano a essere integrati nell'overlay OpenZiti e il meccanismo di autenticazione basato su WebAuthn viene mantenuto. Tuttavia, l'implementazione richiede alcune modifiche a livello di deployment e configurazione, necessarie per adattare il sistema al nuovo contesto.

5.4.1 Architettura ICS

In questa sezione viene descritta l'architettura del sistema OT implementato nel testbed fisico, illustrando i principali componenti che lo costituiscono e il loro ruolo all'interno dell'infrastruttura. Dal punto di vista fisico, tutti i dispositivi sono interconnessi tramite un unico switch di rete, al quale ciascun nodo è collegato mediante connessione Ethernet. Questa configurazione consente di riprodurre una topologia semplice ma rappresentativa, in cui i diversi componenti comunicano attraverso un'infrastruttura condivisa.
L'architettura presenta alcune differenze rispetto alla versione virtuale. In questo contesto, il processo è realizzato mediante due bracci robotici e due nastri trasportatori, sui quali vengono movimentati cubetti colorati. Il primo braccio ha il compito di prelevare i cubetti da un nastro e depositarli sul secondo, realizzando un'operazione di trasferimento tra le due linee. Il secondo braccio introduce invece una logica di selezione: un sensore di colore, posizionato lungo il nastro, consente di identificare i cubetti in transito. Nel caso in cui venga rilevato un cubetto di colore giallo, il braccio provvede a prelevarlo e a depositarlo all'interno di una scatola dedicata; in caso contrario, il cubetto viene ritrasferito sul primo nastro.
Questa configurazione consente di modellare un processo di smistamento tipico dei sistemi industriali, in cui operazioni di movimentazione e di classificazione sono integrate all'interno della stessa linea.

PLC e HMI

Nel testbed fisico, le funzionalità di controllo sono implementate mediante due Raspberry Pi dedicati, su ciascuno dei quali è installato OpenPLC. Non è stato possibile riutilizzare l'approccio containerizzato, in quanto il runtime OpenPLC necessita di un accesso diretto alle interfacce hardware del dispositivo, in particolare ai pin GPIO del Raspberry Pi. L'esecuzione in ambiente Docker avrebbe introdotto limitazioni nell'accesso a tali risorse, rendendo più complessa l'integrazione con i componenti fisici.

Ciascun Raspberry Pi ospita quindi un'istanza indipendente di OpenPLC, che esegue la logica di controllo della linea industriale. I PLC comunicano con i nodi di campo, come i Raspberry Pi associati ai bracci robotici, tramite protocollo Modbus TCP, mantenendo invariata la logica di comunicazione già adottata nel testbed virtuale.

La logica di controllo implementata nel testbed fisico si discosta da quella utilizzata nell'ambiente virtuale, in quanto deve gestire un processo più articolato, che tiene in considerazione la presenza di un nastro trasportatore e di sensori reali. Per questo motivo, il programma PLC è stato adattato per coordinare non solo le operazioni di presa e rilascio, ma anche la movimentazione dei pezzi lungo la linea e la loro selezione in funzione del colore rilevato.

Codice 5.9: Programma PLC in ST del testbed fisico

```
PROGRAM program0
  VAR
    START AT %QX0.0 : BOOL;
    STOP AT %QX0.1 : BOOL;
    RUN AT %QX0.2 : BOOL;
    piece_ready AT %IX0.2 : BOOL;
    is_yellow AT %QX0.8 : BOOL;
    piece_picked AT %QX0.3 : BOOL;
    PICK AT %QX0.4 : BOOL;
    POSE_N AT %QX0.5 : BOOL;
    POSE_B AT %QX0.6 : BOOL;
    picked_yellow AT %QX0.7 : BOOL;
    MOVE_CONVEYOR AT %QX0.9 : BOOL;
```

```
    conveyor_speed AT %MW0 : INT;
  END_VAR
  VAR
    R_TRIG1 : R_TRIG;
  END_VAR

  R_TRIG1(CLK := START);
  RUN := NOT(STOP) AND (RUN OR R_TRIG1.Q);
  PICK := NOT(piece_picked) AND piece_ready AND RUN;
  POSE_N := NOT(picked_yellow) AND piece_picked AND RUN;
  POSE_B := picked_yellow AND piece_picked AND RUN;
  IF is_yellow AND NOT(piece_picked) AND piece_ready AND RUN THEN
    picked_yellow := TRUE; (*set*)
  END_IF;
  IF NOT(is_yellow) AND NOT(piece_picked) AND piece_ready AND RUN THEN
    picked_yellow := FALSE; (*reset*)
  END_IF;
  MOVE_CONVEYOR := NOT(piece_ready);
END_PROGRAM


CONFIGURATION Config0

  RESOURCE Res0 ON PLC
    TASK task0(INTERVAL := T#20ms,PRIORITY := 0);
    PROGRAM instance0 WITH task0 : program0;
  END_RESOURCE
END_CONFIGURATION
```

Rispetto al testbed virtuale, la logica introduce variabili legate al comportamento fisico della linea. La variabile `piece_ready` segnala la presenza di un pezzo in posizione utile per il prelievo, mentre `piece_picked` indica che il cubetto è stato effettivamente afferrato dal braccio. La variabile

`is_yellow` rappresenta invece l'informazione proveniente dal sensore di colore, utilizzata per distinguere i pezzi da deviare verso la scatola da quelli da reinserire nel normale flusso della linea. Un aspetto rilevante riguarda l'utilizzo del tipo `Input` per la variabile `piece_ready` (`%IX0.2`), associata a un sensore di posizione collegato ai pin GPIO del Raspberry Pi su cui è eseguito il runtime OpenPLC. Il sensore rileva la presenza di un cubetto in posizione di prelievo sul nastro trasportatore e aggiorna conseguentemente lo stato della variabile.
La variabile `conveyor_speed`, modificabile dal sistema SCADA, evidenzia la possibilità di controllare anche parametri continui del processo, come la velocità del nastro, tramite un registro Modbus.

Per quanto riguarda l'HMI, non si riscontrano differenze sostanziali rispetto alla versione virtuale. L'interfaccia continua a essere fornita dal sistema OpenPLC e consente il monitoraggio e il controllo dello stato del PLC tramite browser.
La principale differenza riguarda le modalità di deployment: nel testbed fisico, l'installazione e la configurazione vengono gestite tramite Ansible, uno strumento di automazione che consente di eseguire operazioni di provisioning e configurazione su nodi remoti in modo riproducibile. Il processo di configurazione è analogo alla versione Docker dell'ambiente virtuale.

Dispositivi di Campo

Nel seguito vengono descritti i dispositivi che costituiscono il livello più basso della linea industriale dell'architettura OT. Tali componenti comprendono sensori e attuatori direttamente coinvolti nel processo produttivo.

Bracci robotici I bracci presenti nell'infrastruttura sono `Waveshare RoArm-M2-S` [18], ognuno dei quali riceve comandi da un Raspberry Pi 4 tramite comunicazione seriale sulla porta `/dev/ttyUSB0`, attraverso l'invio di messaggi in formato JSON che definiscono giunto, posizione angolare, velocità e accelerazione.

```
{"T":101,"joint":4,"rad":1.60,"spd":0,"acc":3}
```

- `T`: tipo di comando; il valore 101 corrisponde al movimento di un giunto

- `joint`: specifica il giunto da muovere (1 - base, 2 - spalla, 3 - gomito, 4 - pinza)
- `rad`: posizione angolare desiderata del giunto
- `spd`: velocità di movimento; se impostato a 0, la velocità viene gestita dal controllore del braccio
- `acc`: accelerazione del movimento

Analogamente alla versione virtuale, il Raspberry Pi comunica con il PLC tramite Modbus TCP, leggendo periodicamente lo stato dei coil.

L'implementazione dello script di controllo del braccio robotico presenta una struttura concettualmente analoga a quella adottata nel testbed virtuale. Per tale motivo, nel seguito verranno evidenziate esclusivamente le principali differenze introdotte dall'utilizzo del dispositivo fisico.
Il comportamento del braccio è modellato attraverso una classe dedicata, che incapsula le primitive di movimento dei singoli giunti, come la rotazione della base, il controllo della spalla, del gomito e della pinza. A partire da tali primitive vengono costruite operazioni di livello superiore, quali il prelievo del pezzo, il deposito sul nastro e il deposito nel contenitore finale.

```
def lower_shoulder(self):
    rad = 0.66
    T = 101
    command = f'{{"T":{T},"joint":{Joint.SHOULDER.value},"rad":{rad},"spd":{
        self._speed},"acc":{self._acc}}}'
    # {"T":101,"joint":2,"rad":0.66,"spd":0,"acc":3}
    self.send_command(command)
```

Codice 5.10: Comando di movimento di un singolo giunto (spalla)

A partire da tali comandi elementari, lo script costruisce sequenze di operazioni più complesse che implementano le logiche di processo, come il prelievo e il rilascio dei cubetti. In questo modo, il controllo a basso livello, relativo al singolo giunto, viene astratto in operazioni complesse in funzione delle variabili lette dal PLC.

```
start_plc(client)
```

```
while client.open():
    run = read_coil(client,2)
    if run:
        pick = read_coil(client,4)
        pose_conveyor = read_coil(client,5)
        pose_box = read_coil(client,6)

        if pick:
            pick_piece(braccio, client)
        elif pose_conveyor:
            pose_on_conveyor(braccio, client)
        elif pose_box:
            pose_on_box(braccio, client)
    sleep(0.5)
```

Codice 5.11: Ciclo di controllo basato su lettura dei coil Modbus

Sensore di colore Per implementare la logica di selezione dei cubetti è stato utilizzato un sensore di colore `CQRobot TCS34725FN` [4]. Il dispositivo è collegato ai pin del Raspberry Pi associato a uno dei due bracci robotici e consente di rilevare il colore del cubetto in transito, fornendo al sistema di controllo l'informazione necessaria per determinare la successiva operazione di smistamento.
Il sensore è gestito da uno script Python in esecuzione sul Raspberry Pi, che acquisisce periodicamente i valori cromatici rilevati e li traduce in una variabile booleana resa disponibile al PLC tramite protocollo Modbus TCP.

```
def is_yellow(R, G, B) -> bool:
    if R > 150 and G > 150 and B < 120:
        if abs(R - G) < 80:
            return True
    return False

...

        if is_yellow(R, G, B):
```

```
            client.write_single_coil(8, True)
            print("YELLOW")
        else:
            client.write_single_coil(8, False)

...
```

Codice 5.12: Classificazione del colore e aggiornamento del PLC

Nastri trasportatori I nastri trasportatori presenti nella linea sono azionati da motori in corrente continua, pilotati tramite un modulo *DC motor driver*.

Il motore è gestito da un microcontrollore Raspberry Pi Pico, responsabile della generazione dei segnali di controllo per il driver. Il Pico comunica tramite interfaccia seriale con un Raspberry Pi, che svolge il ruolo di nodo intermedio tra il livello di campo e il PLC.

Analogamente agli altri dispositivi dell'infrastruttura, il Raspberry Pi legge periodicamente le variabili di controllo dal PLC tramite protocollo Modbus TCP e, in funzione del loro stato, invia al Raspberry Pi Pico i comandi necessari al pilotaggio del nastro trasportatore.

```
while client.open():
    run = read_coil(client,2)
    if run:
        move = read_coil(client,9)
        speed = get_speed(client)
        if (not moving and move) or speed != current_speed:
            moving = True
            current_speed = speed
            move_conveyor(ser, current_speed)
        elif moving and not move:
            moving = False
            stop_conveyor(ser)
    else:
        stop_conveyor(ser)
    sleep(0.2)
```

Codice 5.13: Controllo del nastro sul Raspberry Pi 4 tramite Modbus e seriale

Il microcontrollore esegue un programma minimale in MicroPython, che riceve comandi in formato JSON e li traduce in operazioni di avvio e arresto del nastro.

```
while True:
    if sys.stdin in select.select([sys.stdin], [], [], 0)[0]:
        command = sys.stdin.readline()
        print("[DEBUG] command: ", command)

        try:
            data = json.loads(command)

            action = data.get("action","").upper()
            if action == "START":
                speed = int(data["speed"])
                direction = data["direction"]
                m.MotorRun('MA', direction, speed)
            elif action == "STOP":
                m.MotorStop('MA')
            else:
                print("unknown action ", action)

            print("waiting command...")
        except json.JSONDecodeError:
            print("ERROR: invalid json ", command)
        except KeyError as e:
            print("ERROR: missing json field ", e)
        except Exception as e:
            print("ERROR: ",e)
    time.sleep(0.01)
```

Codice 5.14: Gestione dei comandi sul Raspberry Pi Pico

Sistema SCADA Il servizio è stato deployato su Raspberry Pi sotto forma di container Docker, riutilizzando direttamente l'immagine e la configurazione definite nel Docker Compose del testbed virtuale. Questo approccio ha consentito di preservare la logica di funzionamento del sistema di

supervisione, garantendo continuità tra l'ambiente simulato e quello fisico.
A causa delle modifiche introdotte nella logica di controllo dei PLC, è stato necessario riconfigurare alcuni *data point* del sistema SCADA, al fine di adattarli alla nuova struttura delle variabili di processo.
Il sistema SCADA continua a comunicare con i PLC tramite protocollo Modbus TCP, consentendo il monitoraggio dello stato della linea e la modifica dei parametri di controllo, come la velocità dei nastri trasportatori.

5.4.2 Integrazione di OpenZiti e WebAuthn

Parte dei componenti dell'infrastruttura di sicurezza è stata deployata sui Raspberry Pi sotto forma di container Docker, senza introdurre modifiche sostanziali rispetto alla versione virtuale. Questo ha consentito di riutilizzare parte delle configurazioni esistenti, semplificando il processo di migrazione verso l'ambiente fisico.

Componenti OpenZiti

Il controller OpenZiti è stato deployato su un Raspberry Pi sotto forma di container Docker, in modo del tutto analogo a quanto realizzato nell'ambiente virtuale, riutilizzando la stessa configurazione.

La principale differenza riguarda invece la gestione dei tunneler. Nella versione virtuale, i tunneler erano eseguiti come container *sidecar*, condividendo lo stack di rete con i servizi applicativi. Questo approccio non è direttamente applicabile nel contesto fisico tra container Docker e host.
Di conseguenza, i tunneler sono stati installati direttamente sui dispositivi fisici come servizio di sistema, tramite il pacchetto `ziti-edge-tunnel` distribuito nei repository ufficiali. In questa configurazione, il tunneler opera come processo locale al nodo, intercettando il traffico applicativo e instradandolo all'interno dell'overlay OpenZiti.
Un'eventuale soluzione alternativa, basata sull'utilizzo di un nodo separato per l'esecuzione dei tunneler, avrebbe comportato la presenza di traffico non cifrato tra il servizio applicativo e il tunneler stesso, compromettendo le proprietà di sicurezza dell'architettura.

I tunneler installati sui dispositivi non partecipano direttamente alla *mesh network* dell'overlay, non contribuendo quindi al piano di routing della rete. Per questo motivo, è stato necessario introdurre

un componente aggiuntivo, rappresentato da un *edge router*, deployato come container Docker su un nodo dedicato. Tale componente è responsabile dell'instradamento del traffico all'interno della mesh OpenZiti, consentendo la comunicazione tra le identità distribuite sui diversi nodi dell'infrastruttura.

```
...

 ziti-router:
   image: ${ZITI_ROUTER_IMAGE:-openziti/ziti-router}
   container_name: ziti-router
   depends_on:
     ziti-router-init-container:
       condition: service_completed_successfully
   user: ${ZIGGY_UID:-2171}
   volumes:
     - ziti-router:/ziti-router
   environment:
     ZITI_CTRL_ADVERTISED_ADDRESS: ${ZITI_CTRL_ADVERTISED_ADDRESS:-ziti-controller}
     ZITI_CTRL_ADVERTISED_PORT: ${ZITI_CTRL_ADVERTISED_PORT:-1280}
     ZITI_ENROLL_TOKEN: ${ZITI_ENROLL_TOKEN:-}
     ZITI_ROUTER_ADVERTISED_ADDRESS: ${ZITI_ROUTER_ADVERTISED_ADDRESS:-ziti-router}
     ZITI_ROUTER_PORT: ${ZITI_ROUTER_PORT:-3022}
     ZITI_ROUTER_MODE: ${ZITI_ROUTER_MODE:-host}
     ZITI_BOOTSTRAP: true
     ZITI_BOOTSTRAP_CONFIG: true
     ZITI_BOOTSTRAP_ENROLLMENT: true
     ZITI_AUTO_RENEW_CERTS: true
     ZITI_ROUTER_TYPE: ${ZITI_ROUTER_TYPE:-edge}
     ZITI_BOOTSTRAP_CONFIG_ARGS:
   command: run config.yml
   ports:
     - ${ZITI_INTERFACE:-0.0.0.0}:${ZITI_ROUTER_PORT:-3022}:${ZITI_ROUTER_PORT:-3022}
   expose:
     - ${ZITI_ROUTER_PORT:-3022}
   restart: unless-stopped
   healthcheck:
```

```
      test:
        - CMD
        - ziti
        - agent
        - stats
      interval: 3s
      timeout: 3s
      retries: 5
      start_period: 15s

  ziti-router-init-container:
    image: busybox
    command: chown -R ${ZIGGY_UID:-2171} /ziti-router
    volumes:
      - ziti-router:/ziti-router

...
```

Le configurazioni dell'overlay OpenZiti, incluse le definizioni dei servizi, delle *service policy* e delle identità, sono state mantenute invariate rispetto alla versione virtuale. Ciò evidenzia come il modello di sicurezza adottato sia indipendente dall'ambiente di esecuzione e possa essere applicato in modo consistente sia in contesti simulati sia in infrastrutture fisiche reali, senza richiedere modifiche alla logica di accesso ai servizi.

Componenti WebAuthn

I componenti dell'infrastruttura di autenticazione, ovvero LoginGateway, UserHandler e database, sono stati deployati come container Docker su Raspberry Pi distinti, senza modifiche rispetto alla versione virtuale. Ciascun servizio è stato integrato nell'overlay OpenZiti tramite un tunneler dedicato installato sul dispositivo, mantenendo le stesse configurazioni dell'ambiente virtuale.

La principale differenza riguarda invece la gestione del reverse proxy. Nella versione virtuale, NGINX era implementato come componente centralizzato all'interno dell'overlay, fungendo da punto di ingresso unico verso i servizi protetti.

Nel testbed fisico, il reverse proxy è stato invece distribuito sui dispositivi che espongono un'interfaccia web, ovvero i PLC e il sistema SCADA. In particolare, su ciascun nodo è eseguito un container NGINX responsabile della protezione dell'interfaccia applicativa locale, la cui configurazione Docker Compose è la stessa utilizzata nell'ambiente virtuale. Il reverse proxy rappresenta l'unico servizio esposto sulla rete del dispositivo, mentre i servizi applicativi rimangono accessibili esclusivamente in locale.

```
location / {
    auth_request /_auth;
    error_page 401 = /_login;

    proxy_pass http://localhost:8080/;
    proxy_redirect ~^https?://[^/]+(/.*)$ https://$http_host$1;

    proxy_http_version 1.1;

    proxy_set_header Host $http_host;
    proxy_set_header X-Forwarded-Host $http_host;
    proxy_set_header X-Forwarded-Port $server_port;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
}
```

In questa configurazione, il reverse proxy non opera più come elemento centrale dell'overlay, ma intercetta localmente le richieste HTTP e le sottopone a verifica di autenticazione, delegando il controllo al servizio LoginGateway. In caso di autenticazione non valida, l'utente viene reindirizzato verso l'interfaccia di login, mentre in caso contrario la richiesta viene inoltrata al servizio applicativo in esecuzione sul nodo.

Un ulteriore aspetto rilevante emerso nel passaggio dal testbed virtuale a quello fisico riguarda i requisiti di esecuzione di *WebAuthn*. L'API Web Authentication è infatti disponibile nei browser soltanto in *secure context*, ossia in contesti considerati affidabili dal punto di vista del trasporto e

dell'origine. In ambiente di sviluppo locale ciò può risultare trasparente nel caso di `localhost`, che i browser trattano generalmente come origine affidabile; nel testbed fisico, invece, i servizi di autenticazione vengono raggiunti tramite nomi host dedicati sulla rete del laboratorio e non possono quindi essere esposti in semplice HTTP.

Per rendere possibile l'utilizzo di *WebAuthn* anche nel testbed fisico, è stato necessario configurare i servizi di autenticazione su HTTPS, generando certificati validi per i rispettivi nomi host e firmandoli mediante una Certification Authority (CA) inserita come fidata nel browser utilizzato per le prove. A tal fine, sono stati utilizzati certificati generati mediante il tool `mkcert` [17], che consente di creare una CA locale e di installarla come fidata nel sistema e nel browser, producendo quindi certificati localmente attendibili per i nomi host impiegati nel testbed. Tale scelta si è rivelata particolarmente adatta a un ambiente sperimentale, nel quale non era possibile fare affidamento su una PKI pubblica o su nomi DNS risolvibili esternamente.

È stato quindi necessario definire esplicitamente i nomi host dei servizi di autenticazione nella risoluzione locale del client, mediante opportune entry nel file `/etc/hosts`, così da rendere coerente l'associazione tra indirizzo IP, nome del servizio e origine HTTPS utilizzata dal browser. In questo modo, i componenti *LoginGateway* e *UserHandler* hanno potuto essere raggiunti tramite nomi host stabili e certificati validi per la relativa origine.

Questa esigenza ha comportato anche una modifica nella modalità di avvio del server applicativo. In particolare, il servizio è stato eseguito mediante `gunicorn` [3], specificando esplicitamente il certificato e la chiave privata associati al nome host del nodo.

```
exec gunicorn \
    --bind 0.0.0.0:5000 \
    --certfile logingateway.rpi.ics.pem \
    --keyfile logingateway.rpi.ics-key.pem \
    --workers 3 \
    --threads 3 \
    --timeout 120 \
    app:app
```

Regole di Firewall

Al fine di garantire che i servizi esposti siano accessibili esclusivamente tramite l'overlay OpenZiti, sono state definite opportune regole di firewall su ciascun dispositivo che espone un'interfaccia all'interno dell'overlay.
In particolare, è stato consentito il traffico in ingresso sulla porta applicativa unicamente dall'interfaccia di loopback, bloccando tutte le altre richieste provenienti dalla rete.

Codice 5.15: Regole di firewall per limitare l'accesso ai servizi PLC, impostate tramite Ansible

```
- name: accept only proxy traffic
  ansible.builtin.iptables:
    chain: INPUT
    protocol: tcp
    destination_port: 502
    in_interface: lo
    jump: ACCEPT
    state: present
  tags: start_secure
- name: block all other traffic
  ansible.builtin.iptables:
    chain: INPUT
    protocol: tcp
    destination_ports:
      - "502"
      - "102"
    jump: DROP
    state: present
  tags: start_secure
```

In questo modo, il servizio risulta raggiungibile esclusivamente tramite il tunneler locale, impedendo qualsiasi accesso diretto dalla rete esterna.
Questo accorgimento risulta essenziale, in quanto l'overlay OpenZiti non interviene sulle comunicazioni di rete preesistenti tra i nodi. In assenza di tali restrizioni, i servizi rimarrebbero comunque

esposti tramite i canali tradizionali, risultando potenzialmente soggetti ad attacchi.

6. Validazione Sperimentale

Il presente capitolo ha l'obiettivo di valutare l'efficacia dell'architettura di sicurezza proposta attraverso una serie di test sperimentali condotti sul testbed fisico.
In particolare, l'analisi si concentra sulla capacità del sistema di resistere a tentativi di accesso non autorizzato, simulando il comportamento di un attaccante operante all'interno della rete non protetta. Questo approccio consente di verificare se le misure adottate, tra cui l'utilizzo dell'overlay OpenZiti e del meccanismo di autenticazione basato su WebAuthn, siano in grado di impedire l'accesso diretto ai servizi e di garantire il rispetto del modello Zero Trust.
I test vengono condotti assumendo che l'attaccante abbia accesso alla rete locale e possa tentare di comunicare direttamente con i dispositivi dell'infrastruttura. In tale scenario, verrà dimostrato come i servizi risultino non raggiungibili attraverso i canali tradizionali e come ogni tentativo di interazione debba necessariamente passare attraverso i meccanismi di autenticazione e autorizzazione previsti.

6.1 Scenario non sicuro

In questa sezione vengono presentate alcune azioni eseguibili da un soggetto non autorizzato che riesca a collegarsi alla rete locale del sistema. L'obiettivo è mostrare come, in assenza delle misure di protezione introdotte successivamente, sia possibile compromettere l'intero processo operativo con relativa facilità.

6.1.1 Scansione della rete

Per valutare la superficie di esposizione, viene eseguita una scansione mediante `Nmap`, con l'obiettivo di identificare gli host attivi e i principali servizi raggiungibili dalla rete locale. Il risultato mostra come le principali interfacce di rete utilizzate dai componenti della linea possano essere rilevate dallo strumento, consentendo di ottenere informazioni utili a caratterizzare il ruolo dei diversi nodi all'interno del sistema.

Un esempio significativo è rappresentato dal nodo `192.168.69.2`, riconducibile a un PLC del testbed. L'output della scansione evidenzia la presenza di più servizi esposti, tra cui, di particolare rilevanza, la porta industriale compatibile con *Modbus TCP* (`502/tcp`) e l'interfaccia web raggiungibile sulla porta `8080/tcp`.

```
Nmap scan report for 192.168.69.2
Host is up (0.00022s latency).
Not shown: 65529 closed ports
PORT      STATE SERVICE       VERSION
22/tcp    open  ssh           OpenSSH 10.0p2 Debian 7 (protocol 2.0)
102/tcp   open  iso-tsap      Siemens S7 PLC
502/tcp   open  mbap?
8080/tcp  open  http-proxy    Werkzeug/2.3.7 Python/3.13.5
8443/tcp  open  ssl/https-alt Werkzeug/2.3.7 Python/3.13.5
```

Anche sul nodo SCADA viene rilevato il servizio web accessibile sulla porta `8080/tcp`, identificato inoltre come server Apache Tomcat. Tale informazione costituisce un ulteriore elemento utile a inferire il ruolo del nodo all'interno dell'architettura, distinguendolo dai componenti di controllo e associandolo più plausibilmente a funzioni di supervisione.

```
Nmap scan report for 192.168.69.9
Host is up (0.00021s latency).
Not shown: 65533 closed ports
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 10.0p2 Debian 7 (protocol 2.0)
8080/tcp open  http    Apache Tomcat 9.0.52
MAC Address: 88:A2:9E:5B:0B:A3 (Unknown)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Questo conferma come, nella configurazione non sicura, i principali componenti del sistema risultino direttamente raggiungibili e identificabili da un host connesso alla stessa rete.

6.1.2 Intercettazione del traffico

Un ulteriore aspetto analizzato riguarda la possibilità di osservare il traffico di rete scambiato tra i dispositivi. A tal fine, sono state acquisite catture di traffico in formato `pcap` direttamente sul PLC, mediante `tcpdump`, durante il normale funzionamento del sistema.

Tale scelta metodologica non è finalizzata a simulare direttamente un attacco di tipo *man-in-the-middle*, bensì a verificare se, qualora un soggetto non autorizzato riuscisse a collocarsi in un punto idoneo di osservazione del traffico, possa plausibilmente ricavare informazioni sulle operazioni effettuate dal sistema di controllo.

In particolare, come mostrato in Figura 6.1, sono state osservate comunicazioni tra il PLC e più nodi del testbed, tra cui `192.168.69.4`, `192.168.69.6` e `192.168.69.9`. Le richieste *Modbus TCP* rilevate comprendono operazioni di lettura dei coil (`0x01`), lettura degli ingressi discreti (`0x02`), lettura degli holding register (`0x03`), scrittura di un singolo coil (`0x05`) e, in almeno un caso, scrittura di un singolo registro (`0x06`).

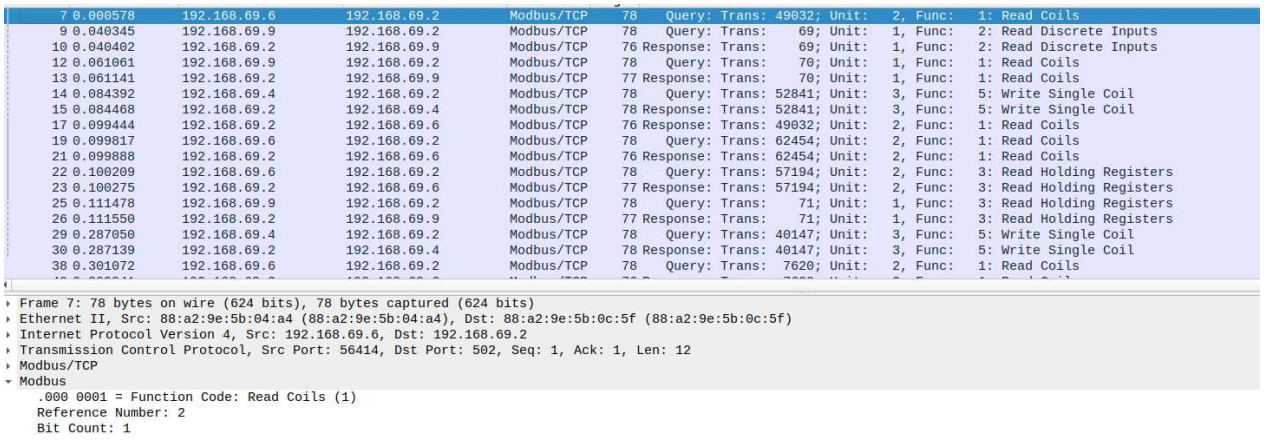

Figura 6.1: Esempio di traffico catturato sul PLC

L'analisi del traffico conferma quindi che, nello scenario non sicuro, la confidenzialità delle comunicazioni non è garantita in modo adeguato. I messaggi applicativi risultano infatti direttamente interpretabili e consentono di osservare non soltanto operazioni di lettura, ma anche operazioni di scrittura verso il PLC. In un contesto OT, tale condizione rappresenta una criticità significativa,

poiché l'osservazione dei messaggi di controllo può costituire un primo passo verso azioni più intrusive, come la replica di richieste legittime o l'invio di comandi non autorizzati.

6.1.3 Interazione non autorizzata con il PLC

Le informazioni ottenute nella fase di ricognizione non hanno un valore puramente descrittivo, ma possono costituire la base per un'interazione diretta con il PLC. In particolare, l'identificazione della porta `502/tcp` come servizio compatibile con *Modbu TCP* indica la presenza di un'interfaccia di comunicazione industriale raggiungibile dalla rete locale e potenzialmente accessibile mediante un client software generico.
Nel contesto del testbed, è stato verificato che un client realizzato in Python, privo di componenti specialistici o proprietari, è sufficiente per instaurare una comunicazione con il PLC ed effettuare operazioni di lettura e scrittura sulle variabili esposte dal protocollo. Questo aspetto è particolarmente rilevante dal punto di vista della sicurezza, poiché mostra come la sola raggiungibilità del servizio consenta, in assenza di ulteriori meccanismi di controllo, un'interazione diretta con il dispositivo.

Per rendere più immediata l'interpretazione degli effetti prodotti dall'interazione non autorizzata con il PLC, viene innanzitutto considerato lo stato del sistema in condizioni nominali di fermo. In tale configurazione, il processo non è in esecuzione e le principali variabili operative assumono valori coerenti con una condizione di inattività.

Point Name	Type	Location	Write	Value
START	BOOL	%QX0.0	true false	FALSE
STOP	BOOL	%QX0.1	true false	FALSE
RUN	BOOL	%QX0.2	true false	FALSE
piece_ready	BOOL	%IX0.2		FALSE
is_yellow	BOOL	%QX0.8	true false	FALSE
piece_picked	BOOL	%QX0.3	true false	FALSE
PICK	BOOL	%QX0.4	true false	FALSE
POSE_N	BOOL	%QX0.5	true false	FALSE
POSE_B	BOOL	%QX0.6	true false	FALSE
picked_yellow	BOOL	%QX0.7	true false	FALSE
MOVE_CONVEJOR	BOOL	%QX0.9	true false	TRUE
convejor_speed	INT	%MW0		0

Figura 6.2: Stato del sistema prima dell'interazione

```
In [1]: from pyModbusTCP.client import ModbusClient

In [2]: client = ModbusClient("192.168.69.2", 502, unit_id=2, timeout=5)

In [3]: client.open()
Out[3]: True

In [4]: client.write_single_coil(0,True)
Out[4]: True

In [5]: client.write_single_coil(1,True)
Out[5]: True

In [6]: client.write_single_coil(2,True)
Out[6]: True

In [7]: client.write_single_coil(3,True)
Out[7]: True

In [8]: client.write_single_coil(4,True)
Out[8]: True
```

Figura 6.3: Scrittura delle variabili mediante un client Python

Point Name	Type	Location	Write	Value
START	BOOL	%QX0.0	true false	TRUE
STOP	BOOL	%QX0.1	true false	TRUE
RUN	BOOL	%QX0.2	true false	FALSE
piece_ready	BOOL	%IX0.2		FALSE
is_yellow	BOOL	%QX0.8	true false	FALSE
piece_picked	BOOL	%QX0.3	true false	TRUE
PICK	BOOL	%QX0.4	true false	FALSE
POSE_N	BOOL	%QX0.5	true false	FALSE
POSE_B	BOOL	%QX0.6	true false	FALSE
picked_yellow	BOOL	%QX0.7	true false	FALSE
MOVE_CONVEJOR	BOOL	%QX0.9	true false	TRUE

Figura 6.4: Stato del sistema successivo all'interazione

La conoscenza degli indirizzi delle variabili di interesse può essere ottenuta in modo relativamente agevole se possibile l'intercettazione del traffico, poiché i messaggi *Modbus TCP* risultano leggibili e consentono di osservare quali coil o registri vengano interrogati o modificati durante il funzionamento del sistema. In tale scenario, un soggetto non autorizzato può ricavare con maggiore precisione quali variabili siano effettivamente coinvolte nella logica operativa del PLC.
Anche in assenza di una preventiva cattura del traffico, l'individuazione di variabili significative può comunque risultare relativamente semplice. Una lettura periodica di insiemi di coil o registri, eseguita durante il normale ciclo di funzionamento, consente infatti di osservare quali valori cambino nel tempo e di associare tali variazioni a specifiche fasi del processo. Sebbene tale procedura non permetta di attribuire immediatamente un significato semantico a ogni variabile, essa è spesso sufficiente a isolare un sottoinsieme di indirizzi rilevanti, sui quali concentrare successive interazioni.

Nel complesso, questa prova mostra come, nello scenario non sicuro, la semplice esposizione del servizio *Modbus TCP* non comporti soltanto una maggiore osservabilità del PLC, ma abiliti anche forme di interazione attiva con il dispositivo. In assenza di autenticazione applicativa, segmentazione efficace e mediazione dell'accesso, un host connesso alla rete locale può quindi non solo identificare il nodo di controllo, ma anche modificare variabili di processo con effetti osservabili sul funzionamento del sistema.

6.1.4 Accesso ai servizi a causa di credenziali non sicure

Un'ulteriore criticità del sistema riguarda la possibilità di accedere alle interfacce web esposte, già individuate nella fase di scansione della rete, mediante l'impiego di credenziali predefinite, vulnerabilità ricorrente nelle infrastrutture OT tradizionali.
Per esempio, collegandosi mediante browser all'indirizzo del PLC sulla porta 8080 precedentemente scoperta, risulta agevole riconoscere il servizio esposto come interfaccia web di *OpenPLC*. La documentazione del progetto indica inoltre che, nella configurazione predefinita, il sistema utilizza credenziali iniziali note, che dovrebbero essere modificate al primo accesso. Tale circostanza rafforza ulteriormente la criticità dello scenario non sicuro, poiché la semplice individuazione del servizio HTTP può tradursi rapidamente in un accesso effettivo all'interfaccia di gestione, qualora le impostazioni iniziali non siano state opportunamente aggiornate.
Lo stesso discorso è valido anche per ScadaBR, da cui è inoltre possibile cambiare la velocità di movimento dei nastri.

6.2 Scenario sicuro

In questa sezione viene verificato sperimentalmente come le attività di osservazione, accesso e interazione non autorizzata, risultate possibili nello scenario non sicuro, non siano più praticabili nella configurazione protetta. L'obiettivo è mostrare l'efficacia delle misure di sicurezza adottate nello scenario sicuro.

6.2.1 Scansione della rete

La scansione della rete nello scenario sicuro mostra una situazione sensibilmente diversa rispetto a quella osservata nella configurazione non protetta. In particolare, alcuni dei servizi direttamente sfruttati nelle prove precedenti non risultano più esposti sulla rete locale, mentre compaiono endpoint riconducibili ai componenti introdotti per la protezione dell'architettura.

```
Nmap scan report for 192.168.69.2
Host is up (0.00033s latency).
Not shown: 65528 closed ports, 2 filtered ports
```

```
Some closed ports may be reported as filtered due to --defeat-rst-ratelimit
PORT      STATE SERVICE       VERSION
22/tcp    open  ssh           OpenSSH 10.0p2 Debian 7 (protocol 2.0)
5050/tcp  open  ssl/http      nginx 1.28.2
8443/tcp  open  ssl/https-alt Werkzeug/2.3.7 Python/3.13.5
```

In particolare, sul nodo PLC non risultano più visibili la porta `502/tcp`, utilizzata nello scenario non sicuro per l'interazione tramite *Modbus TCP*, né la porta `8080/tcp`, che esponeva direttamente l'interfaccia web del dispositivo. Al loro posto compare la porta `5050/tcp`, identificata come servizio `nginx`, coerente con l'introduzione del reverse proxy come punto di accesso mediato ai servizi web.

I risultati ottenuti mostrano quindi una riduzione della superficie di attacco direttamente utile a un soggetto non autorizzato. In particolare, non risultano più immediatamente accessibili alcuni servizi che, nello scenario non sicuro, consentivano l'interazione diretta con i componenti di controllo.

6.2.2 Intercettazione del traffico

Anche nella configurazione sicura è stata effettuata un'acquisizione del traffico di rete direttamente sul PLC, mediante `tcpdump`, con l'obiettivo di verificare se fosse ancora possibile osservare contenuti applicativi interpretabili come nello scenario non protetto.

L'analisi della cattura mostra una situazione significativamente diversa rispetto a quella precedentemente osservata. In particolare, non emergono più messaggi *Modbus TCP* leggibili in chiaro contenenti operazioni di lettura o scrittura. Gran parte dei payload catturati presenta intestazioni *TLS*, riconducibili sia alle fasi di instaurazione della sessione sia al successivo scambio di dati applicativi cifrati. Questo indica che, pur essendo ancora possibile osservare l'esistenza delle comunicazioni tra i nodi, il contenuto informativo dei messaggi non risulta più direttamente accessibile a un osservatore della rete.

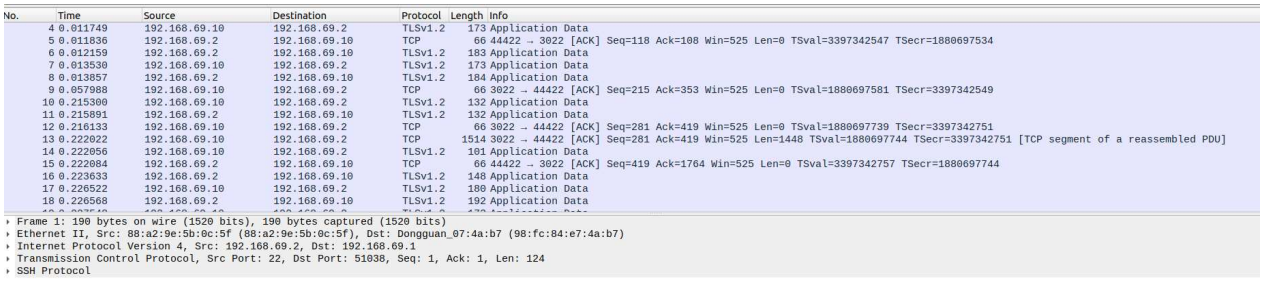

Figura 6.5: Esempio di traffico catturato sul PLC nella configurazione sicura

L'analisi conferma che, nella configurazione sicura, l'osservazione del traffico non consente più di ricavare direttamente informazioni operative sul processo né di identificare con immediatezza le variabili utilizzate dal PLC. La confidenzialità delle comunicazioni risulta quindi rispettata, poiché il traffico applicativo non appare più esposto in forma leggibile sulla rete locale.

6.2.3 Interazione non autorizzata con il PLC

Un'ulteriore verifica sperimentale riguarda il tentativo di ripetere, nella configurazione sicura, la medesima interazione diretta con il PLC che nello scenario non protetto era risultata possibile mediante un semplice client Python. L'obiettivo è verificare se la riduzione della superficie di esposizione osservata nella scansione della rete si traduca effettivamente nell'impossibilità di instaurare una comunicazione applicativa diretta con il dispositivo.

A tal fine, è stato eseguito un tentativo di connessione verso il PLC utilizzando lo stesso approccio già adottato nello scenario non sicuro, ossia mediante un client Python rivolto al servizio *Modbus TCP*. Diversamente da quanto osservato nella configurazione non protetta, il tentativo non ha prodotto una sessione applicativa valida e non ha consentito né la lettura né la scrittura di variabili del PLC.

Figura 6.6: Tentativo di interazione diretta con il PLC mediante client Python nella configurazione sicura

Tale risultato è coerente con quanto emerso nelle prove precedenti. La scansione della rete aveva infatti mostrato la mancata esposizione della porta `502`, mentre l'analisi del traffico non aveva più evidenziato messaggi *Modbus TCP* leggibili in chiaro. Il fallimento della connessione conferma quindi che le modalità di accesso diretto al PLC, risultate praticabili nello scenario non sicuro, non sono più immediatamente replicabili nella configurazione protetta.

Dal punto di vista della sicurezza, questa prova è particolarmente significativa, poiché mostra che la protezione introdotta non si limita a ridurre la visibilità dei servizi, ma impedisce concretamente l'interazione non autorizzata con il nodo di controllo. In questo modo, la sola presenza di un host sulla rete locale non risulta più sufficiente per stabilire una comunicazione diretta con il PLC e modificarne le variabili di processo.

6.2.4 Accesso ai servizi

L'ultima verifica sperimentale riguarda l'accesso ai servizi web nella configurazione sicura. Nello scenario non protetto, l'individuazione delle interfacce HTTP esposte sulla rete locale poteva tradursi rapidamente in un accesso effettivo ai servizi di gestione e supervisione. Nella configurazione protetta, invece, tali interfacce risultano mediate dal reverse proxy e subordinate a un meccanismo di autenticazione passwordless basato su *WebAuthn*.

In particolare, il tentativo di raggiungere mediante browser uno dei servizi precedentemente esposti non conduce più direttamente all'interfaccia applicativa, ma viene intercettato dal componente di

front-end, che reindirizza l'utente verso la procedura di autenticazione. L'accesso al servizio risulta quindi vincolato al possesso di una credenziale precedentemente registrata e associata a un utente autorizzato.

7. Conclusioni

Il presente lavoro ha affrontato il problema della protezione degli accessi e delle comunicazioni in ambito OT. Come mostrato nel corso della tesi, l'evoluzione delle architetture industriali verso modelli sempre più interconnessi, distribuiti e aperti all'integrazione con sistemi IT rende gli approcci di sicurezza tradizionali progressivamente meno adeguati a garantire un livello di protezione coerente con la criticità dei processi controllati.

A partire da queste premesse, la tesi ha dapprima analizzato una linea pilota reale, con l'obiettivo di osservare in che modo le criticità di sicurezza si manifestino in un contesto OT concreto. Tale analisi ha consentito di evidenziare l'esposizione diretta di protocolli industriali e interfacce di supervisione, la possibilità di osservare traffico applicativo non cifrato e la presenza di servizi accessibili mediante credenziali deboli o comunque non adeguatamente protette..

Sulla base delle criticità emerse, è stata quindi progettata una soluzione di sicurezza che combina un overlay di comunicazione fondato su *OpenZiti* e un meccanismo di autenticazione passwordless basato su *WebAuthn*. Tale soluzione è stata implementata dapprima su un testbed virtuale e successivamente su un testbed fisico rappresentativo, costituito da Raspberry Pi, sensori, nastri trasportatori e bracci robotici. Questo doppio livello di implementazione ha consentito, da un lato, di verificare la correttezza e la coerenza dell'architettura in un ambiente controllato e, dall'altro, di valutarne la praticabilità in una configurazione più vicina a uno scenario reale, caratterizzata da vincoli hardware, servizi distribuiti e interazioni fisiche tra i componenti del processo.

La verifica sperimentale ha confermato in modo chiaro l'efficacia delle misure introdotte. Nello scenario non sicuro, le prove hanno mostrato come la scansione della rete consenta di individuare con facilità i principali servizi di controllo e supervisione, come il traffico applicativo risulti osservabile in chiaro e come sia possibile interagire direttamente con il PLC o accedere alle interfacce web mediante credenziali non adeguate. Nello scenario sicuro, al contrario, tali azioni non risultano più immediatamente praticabili: la superficie di attacco osservabile dalla rete locale risulta ridotta, il traffico non espone più contenuti applicativi direttamente interpretabili, l'interazione diretta con il PLC fallisce e l'accesso ai servizi web è subordinato al possesso di una credenziale *WebAuthn* previamente registrata.

Nel complesso, i risultati ottenuti mostrano come l'applicazione congiunta di un modello *Zero Trust* e di un sistema di autenticazione passwordless possa migliorare in modo significativo la protezione di un'architettura OT, senza richiedere la sostituzione dei componenti di processo esistenti. In questo senso, uno dei contributi principali del lavoro consiste nell'aver mostrato la fattibilità concreta di tale approccio non solo in ambiente virtuale, ma anche in un contesto sperimentale fisico.

Tra i possibili sviluppi futuri rientra la possibilità di integrazione di meccanismi di *Intrusion Detection System* (IDS), da impiegare come complemento alle misure di segmentazione logica e autenticazione forte introdotte nella presente tesi. In questo modo, al controllo preventivo degli accessi proprio dell'approccio *Zero Trust* si affiancherebbe una capacità di osservazione e rilevazione di anomalie, utile a identificare comportamenti sospetti anche nel caso in cui essi coinvolgano soggetti o componenti già autorizzati.

Bibliografia

[1] World Wide Web Consortium (W3C). *Web Authentication: An API for accessing Public Key Credentials*. 2021. URL: https://www.w3.org/TR/webauthn/.

[2] FIDO Alliance. *FIDO Alliance Specifications*. URL: https://fidoalliance.org/specifications/.

[3] Paul J. Davis Benoît Chesneaus. *Gunicorn Documentation*. URL: https://gunicorn.org/.

[4] CQRobot. *TCS34725FN Color Sensor SKU: AngelTCS34725US*. URL: http://www.cqrobot.wiki/index.php/TCS34725FN_Color_Sensor_SKU:_AngelTCS34725US.

[5] Descope. *What is WebAuthn?* 2025. URL: https://www.descope.com/learn/post/webauthn.

[6] NetFoundry. *OpenZiti Documentation*. 2024. URL: https://netfoundry.io/docs/openziti/learn/introduction.

[7] Inc. NGINX. *NGINX Documentation*. URL: https://nginx.org/en/docs/.

[8] Liam O Murchu Nicolas Falliere e Eric Chien. *W32.Stuxnet Dossier*. Rapp. tecn. Symantec Security Response, 2011. URL: https://nsarchive2.gwu.edu/NSAEBB/NSAEBB424/docs/Cyber-044.pdf.

[9] Modbus Organization. *Modbus Application Protocol Specification V1.1b3*. 2012. URL: https://www.modbus.org/file/secure/modbusprotocolspecification.pdf.

[10] Modbus Organization. *Modbus Messaging on TCP/IP Implementation Guide V1.0b*. 2006. URL: https://www.modbus.org/file/secure/messagingimplementationguide.pdf.

[11] *Programmable controllers – Part 3: Programming languages*. International Electrotechnical Commission.

[12] OpenPLC Project. *OpenPLC Official Website*. URL: https://www.openplcproject.com/.

[13] ScadaBR Project. *ScadaBR Open Source SCADA*. URL: https://www.scadabr.com.br/.

[14] Stu Mitchell Scott Rose Oliver Borchert e Sean Connelly. *Zero Trust Architecture*. Rapp. tecn. SP 800-207. NIST, 2020. URL: https://doi.org/10.6028/NIST.SP.800-207.

[15] National Institute of Standards e Technology. *Framework for Improving Critical Infrastructure Cybersecurity, Version 2.0*. Rapp. tecn. NIST, 2024. URL: https://doi.org/10.6028/NIST.CSWP.29.

[16] National Institute of Standards e Technology. *Guide to Industrial Control Systems (ICS) Security*. Rapp. tecn. SP 800-82 Revision 3. NIST, 2023. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r3.pdf.

[17] Filippo Valsorda. *mkcert*. URL: https://github.com/FiloSottile/mkcert.

[18] Waveshare. *RoArm-M2-S Robotic Arm Documentation*. URL: https://www.waveshare.com/wiki/RoArm-M2-S.