Relazione su Network Topology Mapper

Cybersecurity Software Architecture and Pattern Design

Team: Mario Mastrulli e Valerio di Maggio

19 dicembre 2024

Per il codice e le instruzioni di installazione, ecco il nostro repository su GitHub: https://github.com/DiMaggioMastrulliCompany/NetworkMapper

Indice

Capitolo 1

Introduzione

Contesto

Nell'era della digitalizzazione, le infrastrutture di rete rivestono un ruolo fondamentale sia nelle attività aziendali che personali, fungendo da pilastri centrali per la comunicazione, la gestione delle informazioni e l'erogazione di servizi critici. Tuttavia, con la crescita esponenziale della complessità e delle dimensioni delle reti moderne, le organizzazioni si trovano ad affrontare sfide crescenti in termini di monitoraggio, gestione e sicurezza.

L'introduzione di tecnologie avanzate come l'**Internet of Things** (IoT), l'espansione delle reti cloud e l'aumento del lavoro remoto hanno ampliato significativamente la superficie di attacco delle infrastrutture digitali. Questo scenario rende la **cybersecurity** una priorità assoluta per proteggere risorse critiche, prevenire attacchi malevoli e garantire la continuità operativa.

In questo contesto, la **mappatura delle topologie di rete** emerge come uno strumento indispensabile per comprendere e monitorare la struttura delle reti. Essa consente di:

- Identificare dispositivi attivi e percorsi di comunicazione;
- Individuare possibili vulnerabilità e punti critici;
- Ottimizzare la gestione delle risorse e migliorare la risposta a eventuali anomalie.

Fornendo una visione chiara e aggiornata dell'infrastruttura di rete, la mappatura facilita la gestione proattiva della sicurezza e migliora l'efficienza complessiva delle operazioni.

Obiettivo del Progetto

Il progetto **Network Topology Mapper** si propone di sviluppare un software in grado di fornire una rappresentazione visiva e organizzata della **topologia di rete**. L'obiettivo principale è offrire uno strumento intuitivo, efficiente e accessibile che permetta agli utenti di comprendere e gestire le reti informatiche.

In particolare, il sistema mira a:

- **Rilevare i dispositivi connessi**: Identificazione di tutti i nodi presenti nella rete, raccogliendo informazioni come indirizzo IP, MAC address, sistema operativo e porte aperte;

- **Individuare le relazioni tra dispositivi**: Creazione di una mappa visuale che mostra i percorsi di comunicazione e le connessioni tra i nodi rilevati;
- **Segnalare potenziali vulnerabilità**: Analisi delle porte aperte e dei servizi attivi per individuare eventuali punti critici o insicurezze;
- **Offrire un'interfaccia user-friendly**: Una piattaforma semplice e intuitiva che consenta a utenti con diversi livelli di competenza di consultare facilmente i dati raccolti, anche grazie al riassunto in linguaggio naturale della rete fornito da un LLM;
- **Supportare decisioni proattive**: Fornire uno strumento per una gestione efficace della sicurezza di rete, anticipando problemi e facilitando interventi tempestivi.

Questo approccio permette di combinare precisione tecnica con un'esperienza utente ottimizzata, rispondendo alle esigenze di professionisti IT e amministratori di rete.

Capitolo 2

Requisiti funzionali e non funzionali

Requisiti Funzionali

I requisiti funzionali definiscono le funzionalità principali che il sistema deve implementare per soddisfare gli obiettivi:

- **Rilevamento dei dispositivi**: Identificazione dei dispositivi connessi alla rete;
- **Generazione della mappa di rete**: Creazione di una rappresentazione visiva e organizzata della topologia di rete;
- **Possibilità di avviare o fermare le scansioni**: Controllo delle operazioni di scansione in tempo reale;
- **Riconoscimento e classificazione dei dispositivi**: Categorizzazione dei nodi rilevati, distinguendo tra dispositivi personali, router, switch, computer, ecc.;
- **Visualizzazione dei dati**: Mostrare i risultati delle scansioni tramite un'interfaccia grafica interattiva;
- **Sommario della rete**: Fornire all'utente un riassunto testuale della topologia della rete, in modo da facilitare utenti di diverso livello di expertise a comprenderne l'architettura;
- **Scansione di dispositivi e sottoreti esterne**: Oltre alla scansione automatica della rete locale, è possibile scansionare qualsiasi indirizzo IP e qualsiasi sottorete raggiungibile dalla macchina. Tutti i router intermedi vengono aggiunti alla topologia rilevata.

Requisiti Non Funzionali

I requisiti non funzionali definiscono le caratteristiche qualitative del sistema per garantire affidabilità, prestazioni e sicurezza:

- **Scansioni rapide ed efficienti**: Ottimizzazione delle prestazioni per gestire grandi reti;
- **Gestione degli errori**: Sistema robusto con gestione degli errori durante le scansioni;

- **Conformità e Standard**: Aderenza agli standard di sicurezza e alle best practices di sviluppo software;
- **Scalabilità**: Capacita di gestire una rete con un numero elevato di dispositivi e di estendere il sistema con nuove funzionalità;
- **Manutenibilità**: Codice modulare e documentato per facilitare aggiornamenti e modifiche;
- **Interfaccia utente intuitiva**: Design accessibile e user-friendly per semplificare l'utilizzo del sistema;
- **Affidabilità**: Funzionamento continuo e garantito anche in presenza di elevato traffico di rete.

Questi requisiti assicurano che il **Network Topology Mapper** non solo fornisca le funzionalità necessarie, ma soddisfi anche criteri di qualità fondamentali per la gestione e la sicurezza delle reti.

Capitolo 3

Stato dell'Arte

La **mappatura delle reti** ha una storia strettamente legata all'evoluzione delle tecnologie di comunicazione e informazione. Nei primi decenni del XX secolo, con l'avvento delle trasmissioni radio e delle reti telefoniche, emerse la necessità di comprendere e rappresentare le connessioni tra dispositivi per ottimizzare le comunicazioni. Con l'introduzione dei calcolatori elettronici e, successivamente, di Internet, la complessità delle reti aumentò esponenzialmente, rendendo indispensabile la mappatura per garantire efficienza e sicurezza.

Importanza della Mappatura delle Reti

La mappatura delle reti riveste un ruolo cruciale per diverse ragioni:

- **Gestione e Manutenzione**: Una mappa dettagliata consente agli amministratori di rete di identificare rapidamente dispositivi, connessioni e potenziali punti critici, facilitando interventi tempestivi in caso di anomalie.
- **Sicurezza**: Conoscere la topologia della rete è fondamentale per individuare vulnerabilità, monitorare accessi non autorizzati e implementare misure di protezione adeguate.
- **Pianificazione**: La mappatura supporta l'espansione e l'ottimizzazione della rete, permettendo una distribuzione efficiente delle risorse e una migliore pianificazione degli aggiornamenti infrastrutturali.
- **Conformità**: Per molte organizzazioni, mantenere una documentazione accurata della rete è essenziale per rispettare normative e standard di settore.

Tool esistenti

Esistono diversi strumenti attualmente utilizzati per la mappatura delle reti, ognuno con caratteristiche specifiche. Di seguito, analizziamo i più rilevanti:

- **Nmap**: è uno degli strumenti più diffusi per la scansione delle reti. Fornisce informazioni dettagliate sui dispositivi connessi, porte aperte e servizi in esecuzione. Tuttavia, la sua interfaccia è principalmente a riga di comando, il che lo rende meno intuitivo per utenti non esperti;

- **Zabbix**: Strumento di monitoraggio delle reti in tempo reale che offre funzionalità di mappatura e alerting. Sebbene potente, la sua configurazione può risultare complessa e richiede risorse significative;
- **SolarWinds**: Una soluzione commerciale avanzata per il monitoraggio delle reti. Offre funzionalità complete di mappatura e reportistica. Tuttavia, il costo elevato rappresenta un limite per molte realtà;
- **Wireshark**: Uno strumento avanzato per l'analisi del traffico di rete. Sebbene utile per un'analisi dettagliata, non fornisce funzionalità immediate di mappatura visiva della rete.

Punti di forza e debolezza

Ogni strumento presenta vantaggi e limiti che influenzano la scelta per specifiche esigenze di rete:

- **Nmap**:
  - *Punti di forza*: Gratuito, flessibile e potente nella scansione di reti;
  - *Debolezze*: Interfaccia poco intuitiva e curva di apprendimento ripida.
- **Zabbix**:
  - *Punti di forza*: Monitoraggio in tempo reale e alerting configurabile;
  - *Debolezze*: Complessità nella configurazione e utilizzo di risorse elevate.
- **SolarWinds**:
  - *Punti di forza*: Funzionalità complete e reportistica dettagliata;
  - *Debolezze*: Costi elevati e accessibilità limitata.
- **Wireshark**:
  - *Punti di forza*: Analisi approfondita del traffico di rete;
  - *Debolezze*: Non adatto per la mappatura automatica delle reti.

Questa analisi evidenzia come un software di nuova concezione, come il **Network Topology Mapper**, possa colmare i limiti esistenti offrendo una soluzione intuitiva, flessibile e accessibile.

Capitolo 4

Metodologia

4.1 Tool scelti

Soluzioni esistenti

Come descritto nel Capitolo 2, gli strumenti esistenti come **Nmap**, **Zabbix**, **SolarWinds** e **Wireshark** offrono soluzioni parziali ma presentano limiti significativi. In particolare:

- Strumenti come **Nmap** sono potenti ma difficili da utilizzare per utenti non esperti;
- Soluzioni commerciali come **SolarWinds** sono costose e non accessibili per piccole e medie realtà;
- Software come **Wireshark** sono focalizzati sull'analisi del traffico e non sulla mappatura della topologia.

Motivazione della soluzione proposta

La nostra soluzione, **Network Topology Mapper**, è progettata per combinare i punti di forza degli strumenti esistenti in un'unica piattaforma che offre:

- Un'interfaccia utente intuitiva e accessibile;
- Potenza di scansione grazie a **Nmap**;
- Visualizzazione chiara e organizzata della topologia di rete;
- Soluzione open-source e facilmente estendibile.

Questo approccio consente di superare i limiti degli strumenti attuali, offrendo un sistema completo, efficiente e scalabile per la mappatura e la gestione delle reti.

4.2 Modello di Processo Prescelto

Metodologia Agile

Per lo sviluppo del **Network Topology Mapper**, è stata adottata la metodologia **Agile**, un approccio che punta sulla flessibilità, sulla capacità di adattarsi rapidamente ai cambiamenti e sulla collaborazione continua all'interno del team.

Il lavoro è organizzato in cicli brevi e ripetuti, detti sprint, durante i quali il team si concentra su piccoli obiettivi ben definiti. Ogni ciclo include diverse fasi:

- **Pianificazione**: Si stabiliscono le priorità e si dividono le attività principali in compiti più gestibili;
- **Sviluppo**: Il team lavora sull'implementazione delle funzionalità decise, mantenendo un ritmo costante e collaborativo;
- **Testing**: Il codice prodotto viene controllato e testato per garantire che risponda agli standard di qualità richiesti;
- **Revisione**: Alla fine di ogni ciclo, si analizzano i progressi raggiunti e si definiscono i passi successivi, integrando eventuali feedback.

Questo approccio permette di rilasciare funzionalità funzionanti in modo incrementale e di migliorare il prodotto in base ai feedback raccolti durante lo sviluppo.

Durante lo sviluppo, abbiamo utilizzato **GitHub Desktop** per effettuare il *push* degli aggiornamenti sul repository Git, facilitando il versionamento del codice e la collaborazione tra i membri del team.

Questo modello di processo consente di garantire un rilascio incrementale delle funzionalità e facilitare la comunicazione e il coordinamento tra i membri del team.

4.3 Tecnologie Principali

Per la realizzazione del **Network Topology Mapper**, sono state selezionate tecnologie moderne che garantiscono un sistema efficiente, scalabile e facilmente manutenibile. Le principali tecnologie adottate includono:

- **Frontend** Il frontend è sviluppato utilizzando **SvelteKit** e **Tailwind CSS**. SvelteKit è stato scelto per la sua leggerezza e per le elevate prestazioni che garantisce nelle applicazioni web moderne. Grazie al rendering ottimizzato e alla sua architettura reattiva, SvelteKit consente di sviluppare in modo semplice un'interfaccia veloce e performante, migliorando l'esperienza utente. L'utilizzo di **Tailwind CSS** offre un approccio *utility-first* alla progettazione dell'interfaccia grafica, permettendo di sviluppare layout puliti, responsive e facilmente personalizzabili con minime risorse di sviluppo.
- **Backend**: Il backend è realizzato in **Python** utilizzando il framework **FastAPI**. FastAPI è stato scelto per la sua capacità di creare API RESTful veloci, sicure e di facile implementazione. Questo framework supporta nativamente operazioni asincrone, migliorando l'efficienza delle operazioni I/O, come le chiamate a strumenti di scansione di rete esterni.

- **Strumento di Scansione**: La scansione della rete è effettuata tramite **Nmap**, uno degli strumenti più potenti e affidabili per il rilevamento e l'analisi dei dispositivi di rete. L'integrazione di **Nmap** con il backend avviene attraverso il modulo *python-nmap*, che permette di eseguire comandi Nmap e di elaborare i risultati in modo programmabile.

- **Architettura API REST**: La comunicazione tra frontend e backend avviene tramite un'architettura **RESTful**. Questo modello di interazione facilita l'integrazione tra i componenti del sistema e assicura una scalabilità ottimale.

- **Libreria di Visualizzazione**: Per rappresentare graficamente la topologia di rete, il sistema utilizza **Cytoscape.js**, una libreria JavaScript ottimizzata per la visualizzazione di grafi complessi. Cytoscape.js consente di generare grafi interattivi, aggiornati in tempo reale e personalizzati in base alle informazioni raccolte.

Questa scelta tecnologica rappresenta un bilanciamento tra potenza, usabilità e accessibilità, consentendo al **Network Topology Mapper** di rilevare reti di qualsiasi dimensione e complessità.

Capitolo 5

Progettazione

Architettura Generale

L'architettura del **Network Topology Mapper** è progettata per essere modulare, efficiente e scalabile, seguendo un approccio basato su microservizi. I componenti principali sono suddivisi in:

- **Frontend**: Realizzato con *SvelteKit* per garantire un'interfaccia utente intuitiva e performante;
- **Backend**: Sviluppato con *FastAPI* in Python, responsabile della gestione delle richieste API, dell'elaborazione dei dati e dell'integrazione con **Nmap**;
- **Strumento di scansione**: **Nmap** viene utilizzato per eseguire scansioni di rete e rilevare dispositivi e servizi;
- **Database**: Utilizzato per archiviare i risultati delle scansioni e le informazioni sulle topologie di rete;
- **API RESTful**: Interfaccia per la comunicazione tra frontend e backend.

Flusso di interazione

Il flusso di interazione tra i componenti principali segue i seguenti passi:

1. L'utente interagisce con l'interfaccia frontend sviluppata in *SvelteKit*;
2. Le richieste dell'utente vengono inviate al backend tramite chiamate API RESTful;
3. Il backend, sviluppato in *FastAPI*, elabora le richieste e invoca **Nmap** per eseguire la scansione della rete;
4. I risultati della scansione vengono elaborati e salvati nel database;
5. Il backend invia i dati elaborati al frontend, che li visualizza in forma grafica e testuale all'utente.

L'importanza dell'architettura MVC

L'architettura **MVC (Model-View-Controller)** rappresenta uno dei paradigmi di design più utilizzati nello sviluppo di applicazioni moderne. Tale architettura suddivide logicamente un'applicazione in tre componenti principali:

- **Model**: La componente responsabile della gestione dei dati e della logica applicativa. Gestisce lo stato dell'applicazione e comunica con il backend per recuperare o aggiornare informazioni.
- **View**: Si occupa della presentazione dei dati all'utente finale. Nel nostro progetto, la *View* è rappresentata dall'interfaccia utente implementata con Svelte e Tailwind CSS.
- **Controller**: Funziona come intermediario tra il *Model* e la *View*. Riceve le richieste dell'utente, elabora i dati tramite il *Model* e aggiorna dinamicamente la *View*.

Applicazione di MVC nel Network Topology Mapper

Nel contesto del progetto **Network Topology Mapper**, l'architettura MVC è stata così declinata:

- **Model**: Implementato nel backend tramite **Python** e **FastAPI**, dove vengono gestite le entità del programma (Node, Hop e Nodes).
- **View**: Realizzata nel frontend con **SvelteKit** e **Cytoscape.js**, fornendo una rappresentazione grafica interattiva e testuale della rete scoperta (grafo dei nodi, sommario della rete e lista dei nodi con tutte le informazioni rilevate).
- **Controller**: Situato nel backend, contiene la logica dell'applicazione e agisce come intermediario tra il frontend e i servizi Nmap. Gestisce le richieste REST API e fornisce i dati elaborati alla View.

Diagramma dell'Architettura

La figura seguente rappresenta l'architettura generale del sistema, mostrando le interazioni tra frontend, backend e strumenti di scansione.

Benefici dell'architettura MVC per il progetto

L'utilizzo dell'architettura MVC nel progetto presenta numerosi vantaggi:

- **Modularità**: Ogni componente è indipendente, facilitando la sostituzione o l'aggiornamento di una parte senza impattare l'intero sistema.
- **Manutenibilità**: La separazione tra logica, dati e interfaccia semplifica l'implementazione di nuove funzionalità.
- **Scalabilità**: Il sistema può essere facilmente esteso aggiungendo nuovi endpoint API o aggiornando la grafica della View.
- **Collaborazione**: Facilita il lavoro di team diversi (backend e frontend) che possono operare in parallelo.

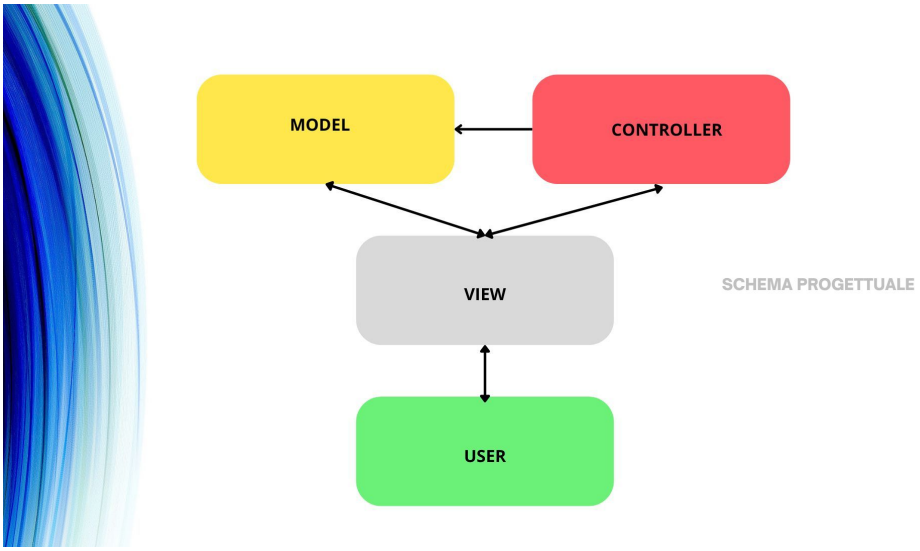

Figura 5.1: Architettura del Network Topology Mapper

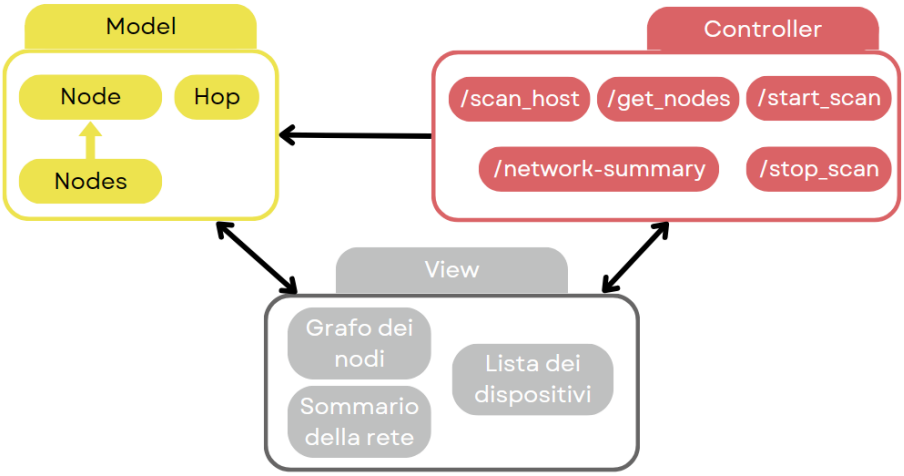

Figura 5.2: Architettura del Network Topology Mapper

Capitolo 6

Implementazione

Architettura Software

Il sistema **Network Topology Mapper** è strutturato in due componenti principali: il **backend** e il **frontend**, che comunicano attraverso un'API REST.

Backend

Il backend, sviluppato in Python utilizzando FastAPI, è il cuore del sistema. Esso si occupa di:

- **Controller principale**: Il *main_controller.py* coordina le operazioni di scansione e aggrega i risultati per l'API REST.
- **Interpretazione delle scansioni**: Il modulo *nmap_wrapper.py* integra lo strumento Nmap per eseguire scansioni di rete dettagliate. Include:
  - Scansione della rete con traceroute per identificare percorsi e dispositivi;
  - Raccolta di informazioni su hostname, OS, porte aperte e MAC address.

  È stato necessario creare una classe wrapper della libreria python-nmap per poter gestire le informazioni sul percorso dei pacchetti (traceroute).
- **Modellazione delle entità di rete**: Tramite i moduli *node.py, hop.py e nodes.py*, viene modellata la struttura dei nodi rilevati con informazioni chiave come indirizzo IP, MAC, open ports e dispositivi collegati.
- **Ottenere informazioni sul localhost**: Il modulo *localhostinfo.py* identifica l'indirizzo IP e il MAC dell'interfaccia di rete locale. Ciò ha necessitato un approccio diverso dagli altri dispositivi in quanto i pacchetti di rete diretti a localhost non venivano indirizzati verso l'esterno, nella rete locale, dunque rendendo più difficile l'identificazione dell'interfaccia di rete usata.
- **Riassumere la topologia di rete**: Il modulo *network_summary.py* permette di ricevere un sommario dell'architettura di rete tramite l'utilizzo dell'Intelligenza Artificiale, in particolare di un Large Language Model.

Frontend

Il frontend è stato realizzato utilizzando *SvelteKit* e *Tailwind CSS*, offrendo un'interfaccia intuitiva per:

- Avviare e interrompere le scansioni tramite pulsanti di controllo;
- Avviare la scansione di un dispositivo o sottorete esterni a quella locale;
- Visualizzare in tempo reale il grafo della topologia della rete utilizzando la libreria **Cytoscape.js**;
- Mostrare dettagli sui nodi rilevati, inclusi IP, OS, porte aperte e distanze hop;
- Presentare un elenco ordinato dei nodi con informazioni complete.
- Fornire un riassunto dell'architettura di rete.

L'interfaccia è progettata per garantire una buona esperienza utente grazie a un layout reattivo agli eventuali cambiamenti rilevati nella rete. L'utente può esplorare facilmente la mappa topologica, identificando visivamente i nodi più importanti e le connessioni della rete.

Interazione tra Componenti

La comunicazione tra frontend e backend avviene tramite le seguenti API REST esposte dal backend:

- **POST /start_scan**: Avvia una nuova scansione;
- **POST /stop_scan**: Interrompe la scansione in corso;
- **POST /scan_host**: Avvia la scansione di un host esterno
- **GET /nodes**: Restituisce la lista aggiornata dei nodi rilevati nella rete.
- **GET /network-summary**: Fornisce il riassunto della topologia sotto forma di stringa, che viene poi renderizzata come Markdown.

Questa architettura garantisce un funzionamento efficiente e scalabile del sistema, consentendo la gestione di reti di diverse dimensioni e complessità.

Implementazione del Backend

Il backend dunque si occupa sia della logica di scansione sia dell'intermediazione con il frontend.

Estratto del Codice: Gestione della Scansione di Rete

Di seguito è riportato un estratto della funzione *scan_network* che si occupa della scansione della rete utilizzando lo strumento **Nmap**, integrato tramite il modulo **nmap-wrapper**.

```
def scan_network():
        # Step 1: Fast ping sweep of the entire subnet
        nm = PortScanner()
        print("Starting fast ping sweep...")
        active_ips = set()

        refresh_network_summary()
        if scan_stop_event.is_set():
            return

        # Initial scan with MAC address detection and traceroute
        nm.scan(hosts="192.168.1.0/24", arguments="-sn --
           traceroute")

        # Process discovered hosts
        for ip in nm.all_hosts():
            if nm[ip].state() != 'down':
                active_ips.add(ip)
                if not nodes.contains_ip(ip):
                    # Get MAC address and vendor from initial
                       scan
                    mac = "unknown"
                    vendor = None
                    if 'addresses' in nm[ip]:
                        mac = nm[ip]['addresses'].get('mac', "
                           unknown")
                        vendor = nm[ip].get('vendor', {}).get(mac
                           )

                    # Create new node
                    node = Node(ip=ip, mac_address=mac, vendor=
                       vendor, status="up", last_seen=datetime.
                       now().isoformat())
                    nodes.add_node(node)

                # Process traceroute data if available
                ...

        print(f"Found {len(active_ips)} active hosts")
```

Listing 6.1: Funzione di Scansione della Rete

Spiegazione del Codice

La funzione *scan_network* è responsabile della scansione continua della rete e raccoglie informazioni sui nodi utilizzando **Nmap**. Ecco una descrizione dei principali passaggi:

- **Creazione dell'istanza Nmap**: Viene inizializzata la classe *PortScanner*, che consente di eseguire comandi Nmap programmaticamente.
- **Scansione rapida** (`-sn --traceroute`): La funzione esegue un ping sweep per rilevare dispositivi attivi nella rete e raccoglie i percorsi di traceroute per ciascun host.
- **Identificazione dei nodi**: Ogni indirizzo IP rilevato viene trasformato in un oggetto **Node**, che include informazioni di base come l'indirizzo IP, lo stato e il timestamp dell'ultima rilevazione.
- **Percorsi di Traceroute**: Se disponibili, i dati dei traceroute (hops intermedi) vengono analizzati e associati al nodo corrispondente utilizzando la classe **Hop**.
- **Gestione delle eccezioni**: Eventuali errori durante la scansione vengono catturati e stampati a console per evitare blocchi nel ciclo.

Integrazione con FastAPI

Questa funzione è invocata all'avvio della scansione tramite l'endpoint:

```
1 @app.post("/start_scan")
2 async def start_scan():
3     global scan_thread
4     if not scan_thread or not scan_thread.is_alive():
5         scan_stop_event.clear()
6         scan_thread = threading.Thread(target=scan_network,
              daemon=True)
7         scan_thread.start()
8         return {"message": "Scanning started"}
```

L'endpoint `/start_scan` avvia la funzione di scansione in un thread separato, garantendo che il backend rimanga responsivo durante l'operazione.

Output della Scansione

Il risultato della scansione è accessibile tramite l'endpoint `/nodes`, che restituisce un elenco di nodi rilevati in formato JSON. Ogni nodo include:

- Indirizzo IP, MAC address e Hostname (se disponibile);
- Sistema operativo rilevato;
- Porte aperte e servizi associati;

Oltre ai dati raccolti sui nodi, l'endpoint `/nodes` è progettato per supportare aggiornamenti incrementali, restituendo i nuovi risultati man mano che la scansione procede. Questo approccio consente di ottimizzare la visualizzazione in tempo reale della topologia di rete senza dover attendere il completamento della scansione.

Salvataggio su database

Ogni volta che l'elenco dei nodi viene modificato, viene effettuato un salvataggio delle informazioni su un database SQLITE3 al percorso *backend_nodes.db*.

Capitolo 7

Testing

Esempio di Scansione

La figura seguente mostra un esempio di topologia di rete generata dal sistema:

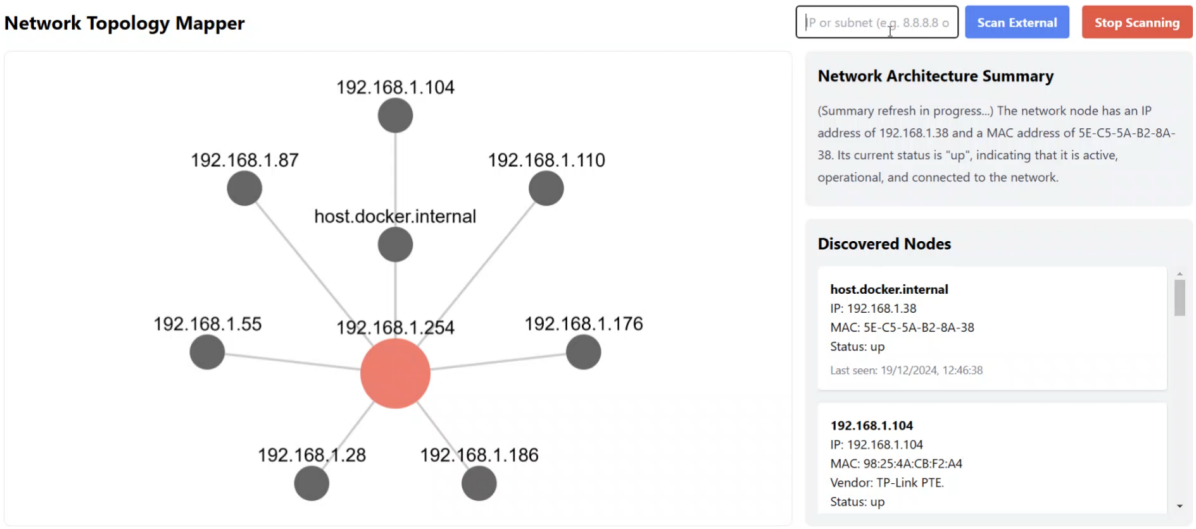

Figura 7.1: Esempio di scansione della rete locale

Da notare in questo esempio che il riassunto non si è ancora aggiornato. Aspettando qualche secondo si ottiene questo:

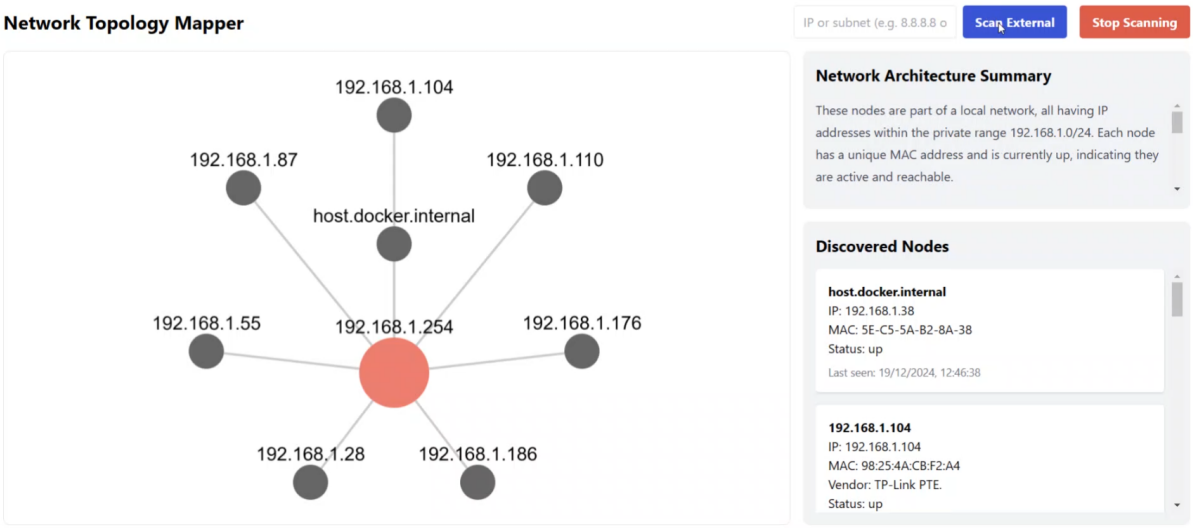

Figura 7.2: Esempio di scansione della rete locale con riassunto generato

Scansione di indirizzo IP esterno

Innanzitutto bisogna scrivere l'IP nel campo apposito, per esempio l'IP del DNS di CloudFlare:

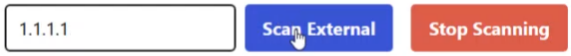

Figura 7.3: Inserimento dell'host esterno

Qualche secondo dopo il programma si aggiornerà mostrando man mano più lontano dalla rete locale i router intermedi e l'IP specificato:

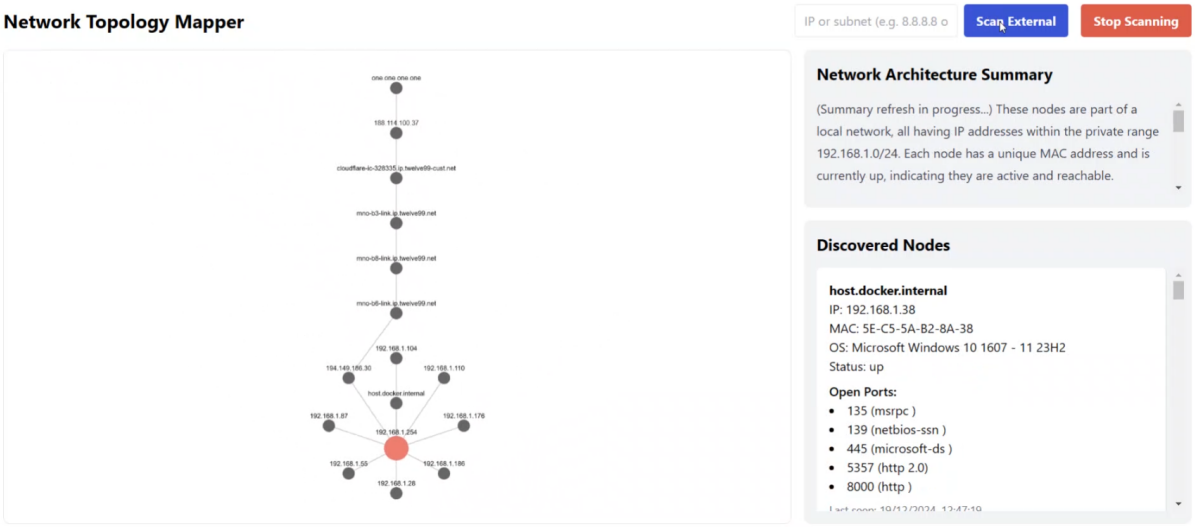

Figura 7.4: Scansione di rete locale e IP 1.1.1.1

Scansione di un secondo indirizzo IP esterno

Proviamo a scansionare un altro indirizzo IP, inserendo nell'apposito campo l'indirizzo IP del DNS di Google. Aspettando qualche secondo si ottiene:

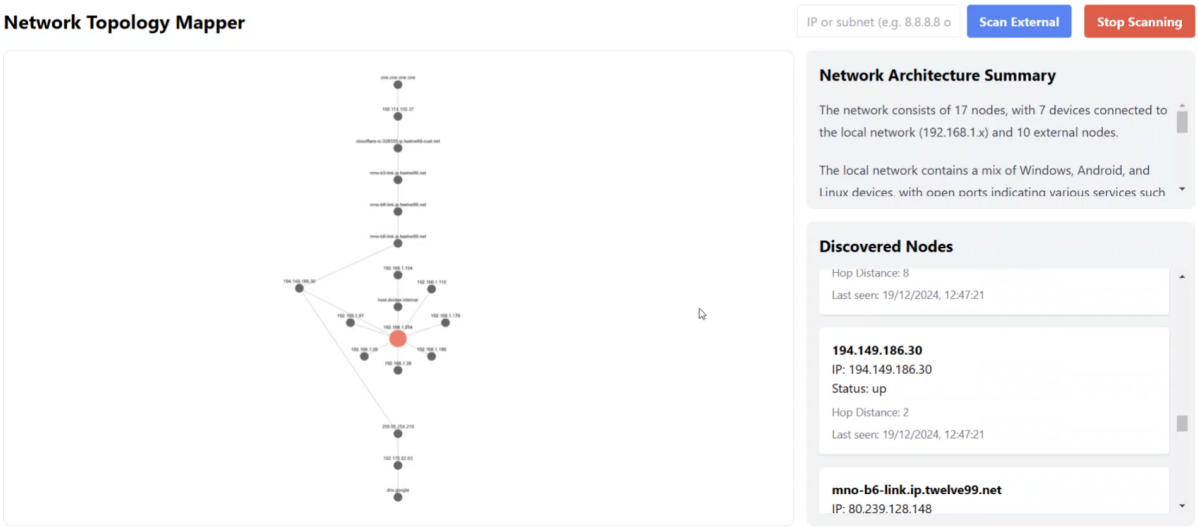

Figura 7.5: Scansione di rete locale, 1.1.1.1 e 8.8.8.8

Come si può notare dall'immagine, si ha un dispositivo (sulla sinistra nel grafo) esterno ma connesso al gateway attraverso cui si instrada il traffico per entrambi gli host scansionati.

Capitolo 8

Conclusione e Future Work

Conclusioni

Il progetto **Network Topology Mapper** ha fornito una soluzione efficace e moderna per la mappatura della topologia di rete. Attraverso l'integrazione di tecnologie avanzate come **Nmap**, **FastAPI** e **SvelteKit**, siamo riusciti a realizzare un sistema in grado di:

- Rilevare rapidamente dispositivi connessi alla rete;
- Visualizzare in tempo reale una mappa interattiva della topologia;
- Fornire dettagli completi su ciascun dispositivo, inclusi indirizzi IP, MAC, sistema operativo e porte aperte;
- Offrire un'interfaccia user-friendly per utenti tecnici e non.

L'implementazione di un'architettura scalabile e modulare permette al sistema di essere facilmente estensibile e adattabile a reti di qualsiasi dimensione, con prestazioni ottimali e tempi di scansione contenuti.

Lezioni Apprese

Durante lo sviluppo del progetto, il team ha affrontato diverse sfide che hanno fornito importanti insegnamenti:

- **Gestione della concorrenza**: L'utilizzo del threading per eseguire le scansioni di rete ha richiesto una gestione attenta delle risorse;
- **Parsing dei dati di Nmap**: L'elaborazione dei risultati XML di Nmap ha evidenziato l'importanza di un'analisi accurata dei dati;
- **Visualizzazione dei dati**: La scelta di *Cytoscape.js* si è rivelata efficace per rappresentare reti complesse in modo chiaro e interattivo;
- **Usabilità**: Ha guidato l'ottimizzazione dell'interfaccia utente, migliorando la navigabilità e l'accessibilità.

Futuri sviluppi

Nonostante i risultati raggiunti, ci sono diverse aree di miglioramento e potenziamento del sistema che possono essere sviluppate in futuro. Tra queste, l'integrazione con altri strumenti di sicurezza come **SNMP** e **Wireshark** consentirebbe un'analisi più approfondita della rete, combinando le informazioni topologiche con dati in tempo reale sul traffico e sugli stati dei dispositivi.

Un'altra direzione fondamentale è l'implementazione del **supporto per analisi storica**, attraverso l'integrazione di un database per memorizzare e confrontare i risultati delle scansioni nel tempo. Questa funzionalità permetterebbe di monitorare l'evoluzione della rete, identificando modifiche significative o comportamenti anomali nel corso del tempo.

Inoltre, il **rilevamento automatico delle vulnerabilità** potrebbe essere realizzato integrando strumenti avanzati di *vulnerability assessment*, come OpenVAS o Nessus. Questo permetterebbe di individuare e segnalare potenziali punti critici nella rete, migliorando così la capacità di prevenire e mitigare le minacce alla sicurezza.

L'implementazione di container **Docker o Kubernetes** potrebbe semplificare il deployment e migliorare la distribuzione su infrastrutture cloud pubbliche o private. L'espansione di queste funzionalità garantirà che il *Network Topology Mapper* continui a essere uno strumento potente e innovativo per il monitoraggio e l'analisi delle reti, adattandosi dinamicamente alle esigenze in continua evoluzione del settore della **cybersecurity**.

Bibliografia

[1] *Nmap Documentation*. Disponibile su `https://nmap.org`.

[2] *FastAPI Documentation*. Disponibile su `https://fastapi.tiangolo.com`.

[3] *SvelteKit Documentation*. Disponibile su `https://kit.svelte.dev`.