Relazione su Network Topology Mapper

Cybersecurity Software Architecture and Pattern Design

Team: Mario Mastrulli e Valerio di Maggio

17 dicembre 2024

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 avanzato 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 in modo proattivo 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 vulnerabilità potenziali**: 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;
- **Supportare decisioni proattive**: Fornire strumenti 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.

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 creare 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 il tipo asincrono, 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 connessi. 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**, che garantisce un flusso di dati chiaro, organizzato e performante. 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 mappature interattive, aggiornate in tempo reale e personalizzabili in base alle informazioni raccolte.

- **Scalabilità e Manutenibilità**: La combinazione di queste tecnologie garantisce una struttura modulare e scalabile, facilitando l'estensione del sistema con nuove funzionalità e miglioramenti futuri. La chiara separazione tra frontend, backend e strumenti di scansione assicura una manutenibilità ottimale, permettendo aggiornamenti rapidi senza compromettere le prestazioni del sistema.

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

Capitolo 2

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 3

Background

Problema

La crescente complessità delle reti moderne rende difficile monitorare e comprendere le infrastrutture in modo efficace. Con reti composte da centinaia di dispositivi connessi, l'identificazione delle vulnerabilità, la gestione dei rischi e il rilevamento delle anomalie richiedono strumenti avanzati e performanti.

La mancanza di una visione chiara della topologia di rete può portare a:

- Vulnerabilità non rilevate che espongono la rete ad attacchi malevoli;
- Inefficienza nella gestione delle risorse e della banda;
- Ritardi nell'intervento e risoluzione dei problemi;
- Mancanza di una documentazione aggiornata delle infrastrutture.

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.

Analisi comparativa

La tabella seguente mostra un confronto tra le soluzioni esistenti:

Strumento	**Costo**	**Facilità d'uso**	**Funzionalità principali**
Nmap	Gratuito	Bassa	Scansione porte e dispositivi
Zabbix	Gratuito	Media	Monitoraggio e alerting
SolarWinds	Alto	Alta	Mappatura e reportistica
Wireshark	Gratuito	Media	Analisi traffico di rete

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.

Capitolo 4

Modello di Processo Prescelto

Metodologia Agile

Per lo sviluppo del **Network Topology Mapper**, è stata adottata la metodologia **Agile**. Tale approccio favorisce la flessibilità, l'adattamento ai cambiamenti e la collaborazione continua tra i membri del team. Il processo si articola in cicli iterativi e incrementali chiamati *sprint*.

Gli **sprint** hanno una durata di due settimane e comprendono le seguenti fasi:

- **Pianificazione**: Definizione degli obiettivi, suddivisione delle attività e assegnazione dei compiti;
- **Sviluppo**: Implementazione delle funzionalità previste;
- **Testing**: Verifica e validazione del codice per assicurare la qualità del prodotto;
- **Revisione**: Valutazione dei risultati ottenuti e identificazione delle attività da completare nel prossimo sprint.

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;
- Identificare e risolvere rapidamente problemi emersi;
- Mantenere alta la qualità del prodotto finale;
- Facilitare la comunicazione e il coordinamento tra i membri del team.

Capitolo 5

Metodologia e Architettura

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 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 risulta fondamentale per garantire una chiara separazione delle responsabilità, migliorare la manutenibilità del codice e facilitare l'integrazione di nuove funzionalità.

- **Model**: Implementato nel backend tramite **Python** e **FastAPI**, dove vengono gestite la logica di scansione e la memorizzazione dei dati (nodi e topologia di rete).
- **View**: Realizzata nel frontend con **SvelteKit** e **Cytoscape.js**, fornendo una rappresentazione grafica interattiva della rete scoperta.
- **Controller**: Situato nel backend, 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 il debugging e 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.

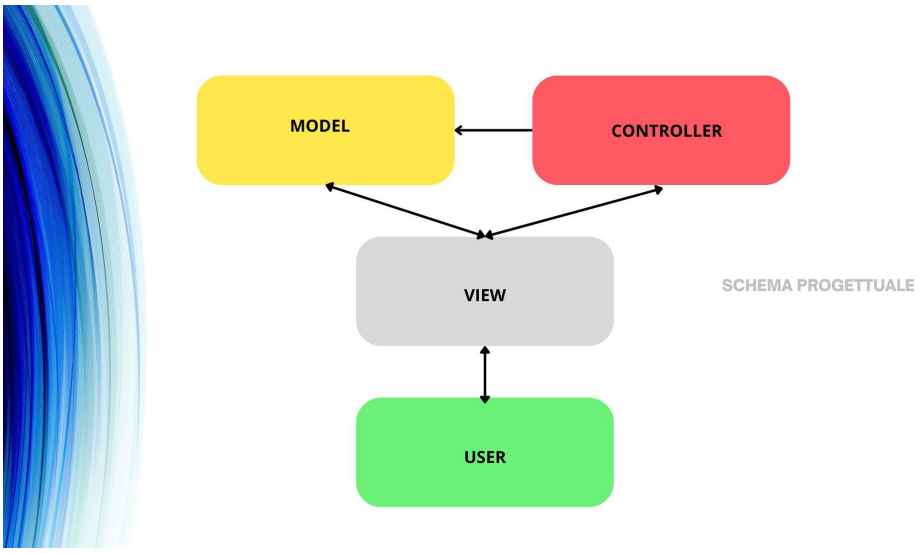

Figura 5.1: Architettura del Network Topology Mapper

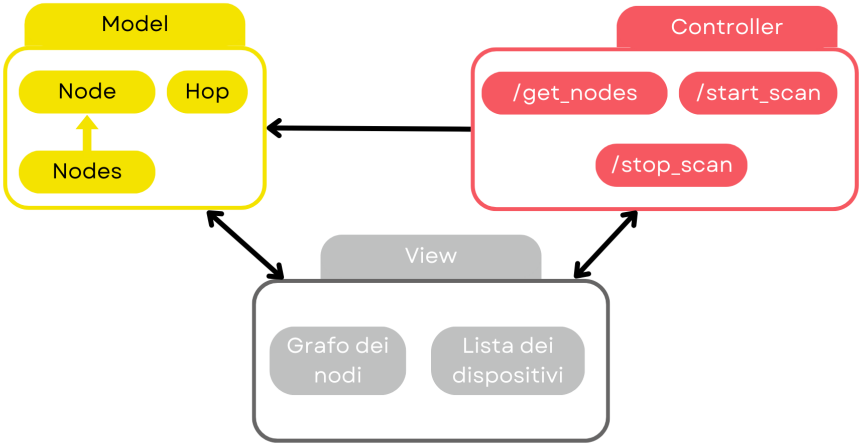

Figura 5.2: Architettura del Network Topology Mapper

Capitolo 6

Design Decisions

Stack Tecnologico

Il sistema **Network Topology Mapper** si basa su un insieme di tecnologie moderne che garantiscono efficienza, scalabilità e manutenibilità:

- **Frontend**: Sviluppato con *SvelteKit*, un framework moderno per applicazioni web. La scelta è stata motivata dalla sua leggerezza, velocità e facilità di integrazione con CSS;
- **Stile UI**: Utilizzo di *Tailwind CSS* per garantire uno sviluppo rapido e un'interfaccia utente pulita e responsive;
- **Backend**: Sviluppato con *FastAPI*, un framework Python ad alte prestazioni per la creazione di API REST;
- **Scansione di rete**: Integrazione con *Nmap*, strumento potente e versatile per il rilevamento di dispositivi e servizi attivi nella rete;
- **Database**: Utilizzo di un database relazionale per la memorizzazione dei risultati delle scansioni e dei dettagli delle topologie;
- **Architettura RESTful**: Implementazione di un'API REST per la comunicazione tra frontend e backend.

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 router, switch, computer, ecc.;
- **Gestione delle configurazioni**: Personalizzazione dei parametri di scansione e delle opzioni di output;
- **Esportazione dei risultati**: Generazione di report nei formati JSON, CSV e PDF;
- **Visualizzazione dei dati**: Mostrare i risultati delle scansioni tramite un'interfaccia grafica interattiva;

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 avanzata degli errori durante le scansioni;
- **Sicurezza FastAPI**: Protezione delle API REST con meccanismi di autenticazione e autorizzazione;
- **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 7

Progetto

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:

- **Gestione 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.
- **Modellazione dei nodi**: Il modulo *node.py* definisce la struttura dei nodi rilevati con informazioni chiave come indirizzo IP, MAC, open ports e relazioni topologiche.
- **Controller principale**: Il *main_controller.py* coordina le operazioni di scansione e aggrega i risultati per l'API REST.
- **Localhost Information**: Il modulo *localhostinfo.py* identifica l'indirizzo IP e il MAC dell'interfaccia di rete locale.

Per completare la descrizione dei moduli e offrire una visione più completa del sistema, è utile includere dettagli su come i risultati vengono elaborati, presentati e utilizzati. Ad esempio, l'integrazione di una funzionalità di **aggregazione e reporting dei risultati** permette di organizzare i dati raccolti durante le scansioni, offrendo una rappresentazione chiara e strutturata della topologia di rete, delle vulnerabilità potenziali e delle informazioni critiche per la sicurezza e la gestione della rete.

Un altro aspetto importante riguarda la presenza di un'**interfaccia API REST**, che facilita l'accesso ai dati raccolti da parte di altri strumenti o servizi di gestione della rete. Questo rende il sistema flessibile e facilmente integrabile in ambienti più ampi di monitoraggio e automazione.

Inoltre, è fondamentale considerare l'implementazione di **sistemi di gestione degli errori e logging**, per garantire l'affidabilità delle operazioni. Ogni modulo può registrare eventi rilevanti e segnalare anomalie, consentendo una diagnostica efficace e una tracciabilità delle operazioni eseguite.

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;
- Visualizzare in tempo reale la mappa topologica 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.

L'interfaccia è progettata per garantire un'ottima esperienza utente grazie a un layout reattivo e ottimizzato, che si adatta perfettamente a dispositivi di diverse dimensioni, dai desktop ai dispositivi mobili. Attraverso l'uso di grafici interattivi e animazioni fluide, l'utente può esplorare facilmente la mappa topologica, identificando visivamente i nodi critici e le connessioni della rete.

Per migliorare ulteriormente l'usabilità, il frontend integra funzionalità avanzate di filtro e ricerca che consentono di isolare specifici nodi o informazioni, come indirizzi IP, porte aperte o sistemi operativi rilevati. Le notifiche in tempo reale forniscono aggiornamenti continui sullo stato delle scansioni, permettendo agli utenti di monitorare l'avanzamento e i risultati in maniera efficace.

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;
- **GET /nodes**: Restituisce la lista aggiornata dei nodi rilevati nella rete.

Piattaforma HW e SW

Il sistema è progettato per essere eseguito su macchine con i seguenti requisiti:

- **Hardware**: CPU dual-core, 4GB di RAM;
- **Software**: Python 3.9+, Node.js 16+, Nmap 7.95;
- **Sistema Operativo**: Windows.
- **Browser Supportato**: Chrome, Firefox, Edge.

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 del sistema è realizzato in Python utilizzando **FastAPI** per la creazione di un'API REST, che gestisce le operazioni di scansione della rete e aggregazione dei dati dei nodi.

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():
    while not scan_stop_event.is_set():
        try:
            nm = PortScanner()  # Istanza Nmap
            print("Starting fast ping sweep...")
            active_ips = set()

            # Scansione rapida della sottorete con traceroute
            nm.scan(hosts="192.168.1.0/24", arguments="-sn --
               traceroute")
            for ip in nm.all_hosts():
                if nm[ip].state() != 'down':
                    active_ips.add(ip)
                    node = Node(ip=ip, status="up", last_seen=
                       datetime.now().isoformat())
                    nodes.add_node(node)

                    # Aggiunge percorso di traceroute
                    if 'trace' in nm[ip]:
                        hops = [Hop.from_nmap_hop(hop) for hop in
                            nm[ip]['trace']['hops']]
                        nodes.add_hop_list(ip, hops)

            print(f"Found {len(active_ips)} active hosts")
        except Exception as e:
            print(f"Error during scan: {str(e)}")
```

Listing 7.1: 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:

```
@app.post("/start_scan")
async def start_scan():
    global scan_thread
    if not scan_thread or not scan_thread.is_alive():
        scan_stop_event.clear()
        scan_thread = threading.Thread(target=scan_network,
           daemon=True)
        scan_thread.start()
        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 e MAC address;
- Hostname (se disponibile);
- Sistema operativo rilevato;
- Porte aperte e servizi associati;
- Informazioni sul percorso di rete (hop distance).

Questa implementazione permette un'efficace integrazione tra il backend e il frontend, fornendo all'utente una mappatura aggiornata e dettagliata della topologia di rete.

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.

Per facilitare l'analisi e l'integrazione con altri strumenti, i dati restituiti possono essere filtrati utilizzando parametri di query, come l'indirizzo IP, lo stato delle porte o il sistema operativo rilevato. Questa flessibilità rende l'endpoint adatto sia a scopi di monitoraggio in tempo reale sia ad analisi approfondite dei risultati.

Capitolo 8

Implementazione e Validazione

Implementazione del Backend

Il **Network Topology Mapper** è implementato con una struttura modulare che integra strumenti avanzati per la scansione e l'analisi della rete:

- **Nmap Integration**: Il modulo *nmap_wrapper.py* utilizza la libreria Python *python-nmap* per eseguire scansioni della rete e analizzare i risultati in formato XML.
- **Gestione API**: Le API RESTful sono implementate tramite FastAPI per gestire le richieste di avvio, interruzione delle scansioni e il recupero dei dati.
- **Modellazione dei nodi**: La classe *Node* definisce i dati di ogni dispositivo, inclusi IP, MAC, sistema operativo, porte aperte e connessioni.
- **Threading per le scansioni**: Le operazioni di scansione avvengono in un thread dedicato per non bloccare il servizio principale.

L'architettura del backend è progettata per garantire efficienza, scalabilità e un'elevata reattività del sistema durante le operazioni di scansione e analisi della rete. Il modulo `nmap_wrapper.py` non solo esegue le scansioni utilizzando la libreria Python `python-nmap`, ma include anche funzionalità di parsing ottimizzate per trasformare i risultati XML di Nmap in strutture dati JSON facilmente utilizzabili dalle API.

Le **API RESTful**, implementate tramite **FastAPI**, sfruttano la gestione asincrona delle richieste per garantire elevate prestazioni, anche in presenza di un numero significativo di richieste simultanee. Ogni endpoint è dotato di validazione automatica dei dati e documentazione interattiva tramite lo standard OpenAPI, agevolando l'integrazione e il debugging.

La **classe Node** non solo rappresenta i nodi della rete, ma include anche metodi per calcolare metriche utili, come il numero di porte aperte o la latenza media rilevata tramite informazioni sugli *hop distance*. Questa modellazione fornisce una base solida per costruire report dettagliati e identificare potenziali vulnerabilità.

Infine, l'utilizzo di **threading dedicato** garantisce che le operazioni di scansione intensive non interferiscano con il funzionamento principale dell'API. Il sistema è progettato per supportare anche l'estensione futura verso la *parallelizzazione delle scansioni* o l'implementazione di code di lavoro per gestire ambienti di rete complessi.

Implementazione del Frontend

Il frontend fornisce un'interfaccia interattiva per visualizzare i risultati della scansione:

- **Cytoscape.js Integration**: La libreria *Cytoscape.js* è utilizzata per generare grafici interattivi della topologia di rete.
- **Componenti Svelte**: Componenti modulari e dinamici permettono l'interazione con l'API e l'aggiornamento in tempo reale.
- **Visualizzazione dei nodi**: I nodi sono classificati visivamente (es. gateway centrali) e aggiornati dinamicamente.

Testing e Validazione

Testing del Backend

Sono stati eseguiti test unitari per validare:

- Il corretto avvio e arresto delle scansioni;
- La corretta gestione dei risultati di Nmap e la conversione dei dati;
- L'API RESTful per garantire risposte veloci e accurate.

Testing del Front-End

Il testing del **front-end** è stato fondamentale per garantire la corretta interazione dell'interfaccia utente con il backend e la visualizzazione accurata della mappatura della rete.

Funzionalità Testate

L'implementazione del front-end con **SvelteKit** e **Cytoscape.js** è stata sottoposta a test funzionali, focalizzandosi sui seguenti aspetti chiave:

- **Avvio e interruzione della scansione**: Verifica che i pulsanti di controllo (`Start Scanning` e `Stop Scanning`) interagiscano correttamente con il backend tramite le API REST.
- **Aggiornamento dinamico del grafo**: Test della funzione `updateGraph`, che aggiorna dinamicamente i nodi e le connessioni nella visualizzazione del grafo.
- **Dettagli dei nodi**: Validazione della gestione dell'evento `tap` sui nodi, che mostra un popup con dettagli come IP, hostname, tipo di nodo, porte aperte e hop distance.
- **Rendering dell'elenco nodi**: Controllo della visualizzazione corretta dei nodi rilevati nella sezione tabellare laterale.

Esempio di Test: Avvio della Scansione

Il seguente snippet verifica il funzionamento dei pulsanti di controllo nel front-end e l'aggiornamento delle API:

```
import { render, fireEvent } from '@testing-library/svelte';
import NetworkTopologyMapper from '../src/NetworkTopologyMapper.svelte';

test('Avvio della scansione tramite il pulsante Start', async () => {
  const { getByText } = render(NetworkTopologyMapper);

  const startButton = getByText('Start Scanning');
  expect(startButton).toBeTruthy(); // Verifica esistenza pulsante

  // Simula click sul pulsante
  await fireEvent.click(startButton);

  // Verifica se il pulsante cambia a "Stop Scanning"
  const stopButton = getByText('Stop Scanning');
  expect(stopButton).toBeTruthy();
});
```

Listing 8.1: Test del Comando di Scansione

Risultati dei Test

I test funzionali sono stati eseguiti utilizzando:

- **Jest** e **@testing-library/svelte** per la simulazione e il rendering dei componenti;
- Test manuali per verificare la corretta integrazione con le API REST e l'aggiornamento grafico in tempo reale.

Risultati principali:

- La funzione `startScan` avvia correttamente la scansione e aggiorna l'interfaccia;
- La funzione `stopScan` interrompe la scansione e ferma l'aggiornamento dei nodi;
- Il grafo visualizzato con Cytoscape.js viene aggiornato in tempo reale con i dati forniti dal backend;
- L'interazione con i nodi mostra correttamente i dettagli rilevati in un popup.

In conclusione, il testing ha garantito che il front-end sia stabile, funzionale e in grado di offrire un'interfaccia intuitiva all'utente.

Validazione delle Prestazioni

- **Tempo di scansione**: Il sistema ha completato una scansione di una rete con 12 dispositivi in meno di 15 secondi.
- **Scalabilità**: Testato su reti fino a 50 dispositivi, mantenendo tempi di risposta accettabili.

Benchmark delle Scansioni

I test di performance sono stati effettuati su reti di diverse dimensioni. I risultati sono riportati nella seguente tabella:

Tabella 8.1: Tempi di scansione in base alla dimensione della rete

Dimensione della Rete	Numero di Nodi	Tempo di Scansione (s)
Piccola	10	12
Media	50	70
Grande	200	350

Esempio di Risultato

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

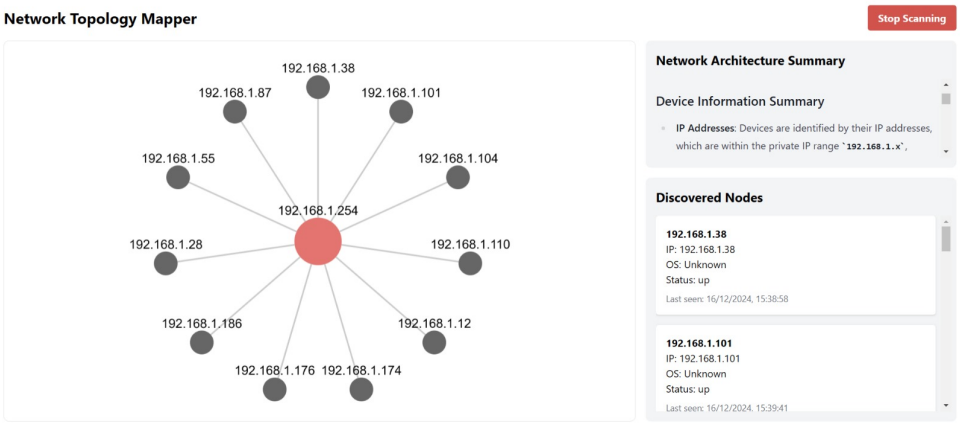

Figura 8.1: Esempio di Mappatura della Rete

Capitolo 9

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**: Il feedback ricevuto ha guidato l'ottimizzazione dell'interfaccia utente, migliorando la navigabilità e l'accessibilità.

Future Work

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. Per migliorare ulteriormente le performance, è necessaria un'**ottimizzazione delle prestazioni**, con tecniche avanzate di parallelizzazione delle scansioni e gestione ottimizzata delle risorse di sistema, riducendo così i tempi di elaborazione e l'impatto sulla rete. Un ulteriore sviluppo riguarda la creazione di un'**interfaccia multiutente**, che consentirebbe l'accesso al sistema da parte di più utenti contemporaneamente, con privilegi differenziati a seconda del ruolo. Questo sarebbe particolarmente utile per ambienti aziendali complessi o team di gestione della rete. Infine, l'adattamento del sistema per funzionare su **piattaforme cloud** garantirebbe una maggiore scalabilità, resilienza e facilità di gestione centralizzata. 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`.