Università Politecnica delle Marche

Facoltà di Ingegneria

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

---

Implementazione di un sistema Zero Trust per un contesto bancario

Docenti

Prof. Luca Spalazzi

Componenti del gruppo

Bellante Luca
Coccia Giansimone
Ferretti Laura
Staffolani Federico
Zazzarini Micol

Anno Accademico 2024-2025

Indice

Elenco degli Snippet di Codice

Elenco delle figure

1 Introduzione

Il progetto si propone di realizzare e implementare una infrastruttura di sicurezza informatica per un sistema bancario, ispirata ai principi dell'architettura Zero Trust, su una rete articolata in tre segmenti, interna, esterna e Wi-Fi, con l'intento di monitorare e regolare in modo rigoroso tutto il traffico diretto al database, risorsa critica dell'infrastruttura. In un contesto in cui le minacce informatiche sono sempre più sofisticate e pervasive, il modello Zero Trust rappresenta un cambio di paradigma rispetto alla sicurezza tradizionale, basata sulla difesa del perimetro. Nel modello Zero Trust infatti, non si concede alcuna fiducia implicita: utenti, dispositivi e applicazioni devono essere autenticati, autorizzati e verificati continuamente, indipendentemente dalla loro posizione o appartenenza alla rete aziendale (Figura 1). In dettaglio, la nostra architettura Zero Trust si fonda sui seguenti concetti chiave:

- **Never trust, always verify**: nessuna entità è considerata affidabile fino a quando non supera verifiche continue di identità, integrità del dispositivo e contesto della richiesta.
- **Principio del minimo privilegio**: ogni accesso è limitato allo stretto necessario, con privilegi concessi dinamicamente.
- **Assume breach e monitoraggio continuo**: si presuppone la compromissione, con analisi in tempo reale del traffico e logging per rilevare movimenti sospetti verso il database.

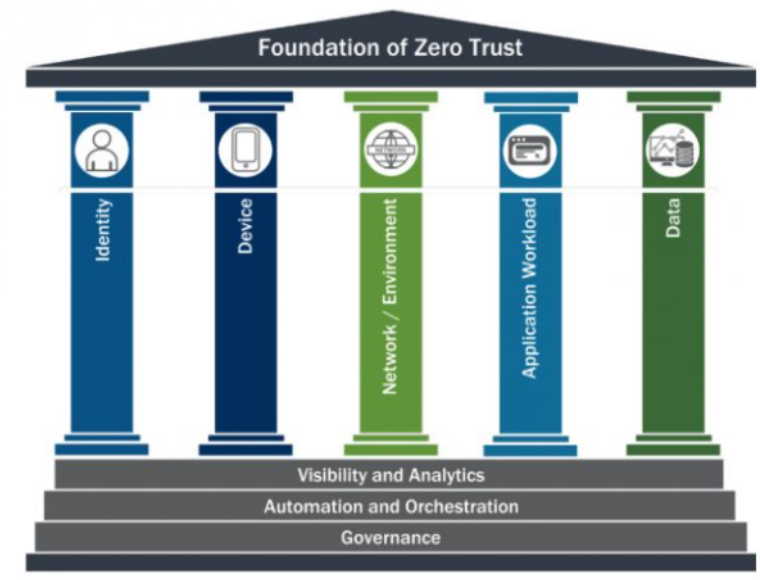

Figura 1: Fondamenti dell'Architettura Zero Trust

Ogni richiesta di accesso alle risorse viene quindi sottoposta a rigorosi controlli di autenticazione, autorizzazione e monitoraggio, con una valutazione dinamica della fiducia basata su molteplici fattori. L'intento del progetto è stato quindi quello di costruire

un'infrastruttura adattiva, resiliente e realmente perimetrale, in cui il database viene protetto da meccanismi quali i controlli di accesso rigorosi e dinamici, il monitoraggio continuo e in tempo reale, l'isolamento efficace mediante micro-segmentazione, la verifica costante della fiducia e la reazione rapida a qualsiasi anomalia. Non quindi una semplice difesa a "castello", ma un approccio strategico che tratta ogni richiesta al database come sospetta fino a prova contraria, garantendo una sicurezza mirata e moderna, capace di rispondere efficacemente alle minacce moderne e di proteggere le informazioni più sensibili dell'organizzazione.

2 Tecnologie utilizzate

Per implementare i principi Zero Trust e garantire un controllo capillare su ogni aspetto della sicurezza, il progetto si avvale di una serie di tecnologie open source e strumenti specifici, ciascuno con un ruolo ben definito sia dal punto di vista teorico che pratico.

2.1 IpTables

IpTables (Figura 2, [5]) è uno strumento fondamentale per la gestione del firewall nei sistemi Linux. A livello teorico, rappresenta un'interfaccia per la configurazione delle regole di filtraggio dei pacchetti di rete, consentendo di definire quali tipi di traffico sono ammessi, bloccati o reindirizzati tra le diverse interfacce di rete. Nel nostro progetto, IpTables viene utilizzato per implementare la segmentazione della rete e applicare policy di accesso rigorose tra i vari segmenti (interna, esterna, Wi-Fi) e i servizi esposti dal gateway. Le regole vengono configurate in modo dinamico tramite script dedicati, permettendo di bloccare indirizzi IP malevoli identificati dagli altri strumenti di sicurezza, limitare la superficie di attacco e isolare eventuali compromissioni. Inoltre, l'integrazione con gli altri componenti consente di aggiornare automaticamente le blacklist e adattare le regole di filtraggio in risposta a comportamenti sospetti rilevati in tempo reale.

Figura 2: IPtables

2.2 Squid

Squid (Figura 3, [4]) è un proxy server caching che, dal punto di vista teorico, funge da intermediario tra i client e i server web, gestendo e ottimizzando le richieste HTTP e HTTPS. Oltre a migliorare le prestazioni grazie alla memorizzazione nella cache delle risorse più richieste, Squid permette di applicare controlli di accesso, autenticazione e filtraggio dei contenuti. Nel contesto del progetto, Squid viene configurato come proxy trasparente all'interno del container gateway, intercettando e filtrando il traffico web proveniente dai diversi segmenti della rete. Questo consente di monitorare le attività degli utenti, applicare policy di accesso basate su ruoli e contesto, e integrare i log generati con gli altri strumenti di sicurezza come Splunk.

2.3 Snort

Snort (Figura 4, [2]) è uno dei sistemi di intrusion detection e prevention (IDS/IPS) open source più diffusi. Snort analizza il traffico di rete in tempo reale, confrontandolo con un

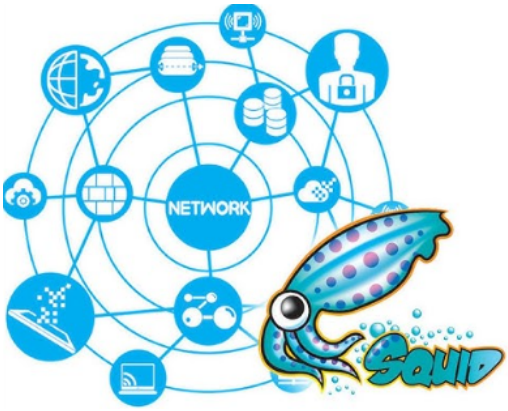

Figura 3: Squid

insieme di regole per identificare pattern riconducibili ad attacchi noti, come tentativi di port scanning, exploit, iniezioni di codice o attacchi DoS. Nel nostro progetto, Snort è installato e avviato su tutte le interfacce del gateway, dove monitora costantemente il traffico tra i vari segmenti della rete. Quando viene rilevata un'attività sospetta o malevola, Snort genera alert e log dettagliati che vengono poi inviati a Splunk e al PDP (Policy Decision Point) per l'aggiornamento dinamico della reputazione degli indirizzi IP coinvolti. In caso di attacchi gravi, il sistema può bloccare automaticamente l'IP sorgente tramite l'aggiornamento delle regole di IpTables, rendendo la risposta agli incidenti tempestiva e automatizzata.

Figura 4: Snort

2.4 Splunk

Splunk (Figura 5, [3]) è una piattaforma avanzata per la raccolta, l'analisi e la visualizzazione dei log e degli eventi generati dai vari componenti dell'infrastruttura. Dal punto di vista teorico, Splunk consente di centralizzare i dati provenienti da fonti eterogenee, facilitando la correlazione degli eventi, l'individuazione di pattern anomali e la generazione di alert in tempo reale. Nel progetto, Splunk riceve e aggrega i log prodotti da Squid e Snort, offrendo una visibilità completa sulle attività di rete e sugli accessi alle

risorse. Grazie alle sue capacità di analisi avanzata, Splunk permette di implementare policy di risposta automatica, come l'aggiornamento delle blacklist o la modifica dei punteggi di fiducia, e di produrre report dettagliati sullo stato di sicurezza della rete.

Figura 5: Splunk

2.5 Docker

Docker (Figura 6, [1]) è una piattaforma di *containerizzazione* che consente di eseguire applicazioni in ambienti isolati, portabili e riproducibili, sfruttando la virtualizzazione a livello di sistema operativo mediante `cgroups` e `namespaces`. I container sono unità leggere che includono file-system, processi e dipendenze, pur condividendo il kernel dell'host, evitando l'overhead associato alle macchine virtuali tradizionali.
Un *Dockerfile* è un file testuale che descrive passo passo come costruire l'immagine container, usando istruzioni come `FROM`, `RUN`, `COPY`, `CMD`, permettendo di ottenere immagini versionabili e modulari attraverso layer.
Docker Compose è lo strumento utilizzato per orchestrare ambienti multi-container. In un file `docker-compose.yml` si definiscono servizi, reti e volumi, abilitando il provisioning, l'avvio e l'arresto di interi stack con un singolo comando.
Nel nostro progetto, abbiamo usato i container non solo come ambienti isolati, ma anche come una *simulazione di molteplici macchine virtuali*: ogni servizio – client, gateway, database, PEP, PDP – è eseguito in un container separato, replicando il comportamento di host distinti senza il peso computazionale delle VM tradizionali.

I benefici principali sono i seguenti:

- **Isolamento e segmentazione**: ogni servizio opera in una "mini-macchina", permettendo controllo e policy granulari in linea con la filosofia Zero Trust.
- **Coerenza tra ambienti**: lo stesso Dockerfile è usato in sviluppo, test e produzione, eliminando difformità nelle dipendenze.
- **Deployment e scalabilità**: Docker-Compose consente di avviare, scalare o riportare online interi stack con facilità.
- **Sicurezza containerizzata**: il traffico tra container è ben definito, monitorabile e ispezionabile a livello Layer 7[1], proteggendo l'interazione col database.

[1]Settimo livello del modello OSI, chiamato *Application Layer*; esso gestisce protocolli applicativi (HTTP, HTTPS, ecc), contenuti e dati delle richieste e identità dell'utente o del servizio.

- **Sostituzione economica delle VM**: i container simulano ambienti distribuiti con tempi e costi ridotti rispetto alle VM tradizionali.

Dunque, grazie a Docker, ai Dockerfile e a Docker Compose, abbiamo potuto definire servizi isolati in container, orchestrare l'intero sistema come su molteplici "macchine virtuali", garantire coerenza, sicurezza e conformità ai principi Zero Trust, con particolare attenzione all'interazione verso il **database**.

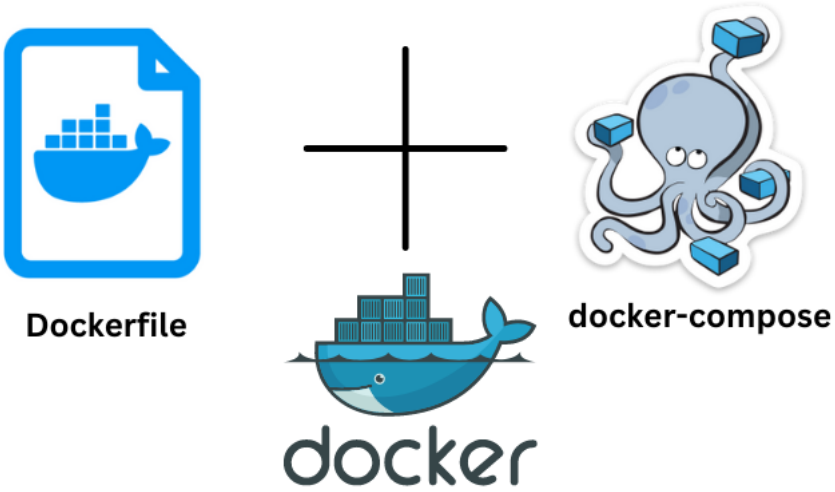

Figura 6: Containerization

3 Architettura

Come anticipato, l'architettura del progetto (Figura 7) si basa su un approccio a microservizi, in cui ogni componente principale dell'infrastruttura di sicurezza è isolato all'interno di un container Docker dedicato. Questo consente di ottenere una segmentazione logica e fisica delle funzioni di rete, migliorando la sicurezza, la scalabilità e la manutenibilità del sistema. Il funzionamento generale prevede che i client (external, internal, wifi), ognuno sulla propria rete, interna, esterna e Wi-Fi rispettivamente, interagiscano con le risorse protette attraverso un Policy Enforcement Point (PEP). Il traffico dai client al PEP viene monitorato dal gateway centrale, che svolge il ruolo di punto di controllo, applicando regole di firewall tramite iptables, gestendo il proxy Squid per il traffico HTTP/HTTPS e monitorando il traffico di rete con Snort. Tutte le richieste di accesso alle risorse vengono inoltrate dal gateway al Policy Enforcement Point (PEP), che si occupa di autenticare l'utente e di inviare una richiesta di autorizzazione al Policy Decision Point (PDP). Il PDP valuta la richiesta in base a delle policy dinamiche, punteggi di fiducia, blacklist e regole di accesso granulari, restituendo una decisione finale al PEP. Se l'accesso è consentito, il PEP esegue l'operazione richiesta sul database, altrimenti la nega. Tutti gli eventi e i log generati dai vari componenti vengono inoltre raccolti e analizzati da Splunk, che fornisce visibilità centralizzata e strumenti di risposta automatica agli incidenti, attraverso una serie di allarmi che si attivano in risposta a determinati eventi rilevati nello storico dei logs. Il database PostgreSQL, protetto tramite TLS, rappresenta la risorsa sensibile a cui si accede solo dopo aver superato tutti i controlli previsti dall'architettura Zero Trust.

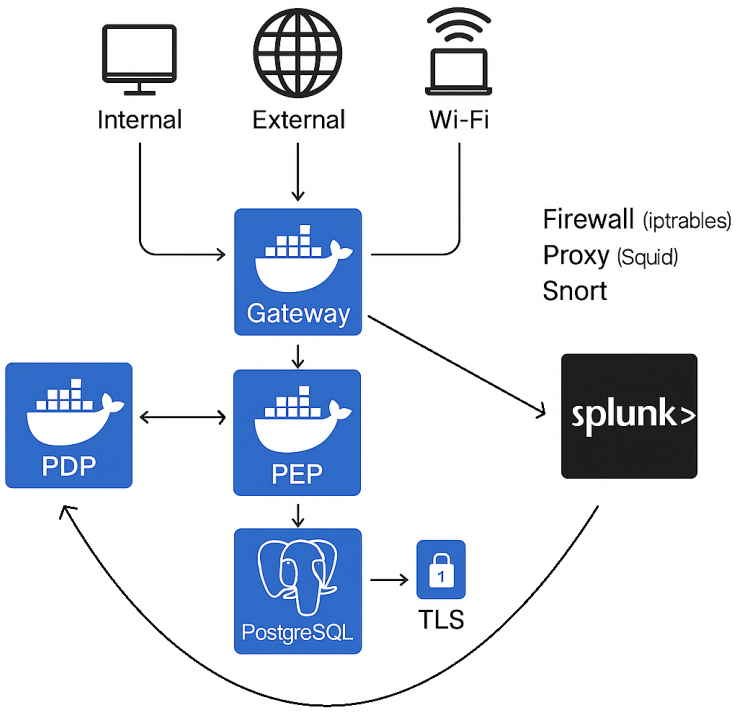

Figura 7: Architettura di sistema

3.1 Servizi

La struttura dei containers all'interno dell'architettura riflette la suddivisione logica delle funzioni di sicurezza e gestione della rete.
Più in dettaglio, i client sono suddivisi in tre containers distinti: `client_internal`, `client_external` e `client_wifi`, ciascuno collegato alla propria rete separata, a simulare diversi segmenti di rete aziendale.
Il container `gateway` rappresenta il nodo centrale di smistamento e controllo, dotato di più interfacce di rete per collegarsi a tutte le subnet, e contenente **iptables**, **Squid** e **Snort**.
Il container `pep` (*Policy Enforcement Point*) è responsabile dell'autenticazione e dell'inoltro delle richieste di autorizzazione al container `pdp` (*Policy Decision Point*), che implementa la logica di decisione basata su policy, reputazione e blacklist.
Il database `db` utilizza un'immagine *PostgreSQL*, e viene inizializzato con script SQL e certificati TLS per garantire la sicurezza delle connessioni.
Infine, il container `splunk` raccoglie e analizza i logs provenienti da Snort e Squid, offrendo funzionalità di monitoraggio avanzato e dashboard personalizzate.
Tutti i containers sono collegati tramite reti Docker bridge dedicate, definite nel file `docker-compose.yml`, che garantiscono isolamento e controllo granulare del traffico tra i vari segmenti dell'infrastruttura.

3.1.1 Client Containers

Come anticipato, all'interno dell'architettura sono stati definiti tre container **client**, ciascuno connesso a una rete virtuale distinta per simulare l'accesso da segmenti di rete differenti:

- `client_internal` connesso a `internal-net` (`172.20.0.2`);
- `client_external` connesso a `external-net` (`172.21.0.2`);
- `client_wifi` connesso a `wifi-net` (`172.22.0.2`).

Tutti i container client sono basati su un'immagine `Ubuntu 22.04`, arricchita con strumenti di rete utili per test e simulazioni, come `curl`, `nmap`, `hping3`, `jq`, e `snort`.
Ogni client ha il proprio gateway di default configurato per indirizzare il traffico verso il container `gateway` (Listing 1), il quale svolge le funzioni di **firewall**, **proxy** e **NIDS** (Snort).

Listing 1: Snippet di configurazione routing per external client

```
1 # Imposta default gateway all'IP del router sulla rete external-net
2 ip route del default 2>/dev/null
3 ip route add default via 172.21.0.3
```

Sono inoltre dotati di uno script interattivo (`uc.sh`) che consente di simulare operazioni applicative verso la risorsa protetta. Più in dettaglio, lo script include una semplice interfaccia testuale per selezionare le caratteristiche dell'operazione e si appoggia al tool `jq` per il parsing dei dati JSON restituiti.
All'avvio, i client eseguono uno script `entrypoint.sh`, che configura la route di default e lancia una shell interattiva.
Questo approccio consente agli operatori di interagire in tempo reale con il sistema e testare le politiche di accesso definite nel `PEP/PDP`.

3.1.2 Container Gateway

Il container `gateway` rappresenta il cuore dell'architettura di sicurezza del sistema, fungendo da **punto di passaggio obbligato** per tutto il traffico proveniente dai clients, e diretto verso il PEP. Il suo ruolo principale è quello di monitorare, filtrare e analizzare il traffico di rete, attraverso diversi strumenti: **Squid** (proxy trasparente), **Snort** (NIDS - Network Intrusion Detection System), **iptables** (firewall e routing), e un meccanismo automatizzato di **blacklist dinamica**.

Il container è costruito a partire da un'immagine Debian, con gli strumenti necessari installati tramite apt e pip:

- **Squid**: per l'intercettazione del traffico HTTP e la sua registrazione.
- **Snort**: per l'analisi in tempo reale del traffico sospetto.
- **iptables**: per la definizione delle regole di routing e di sicurezza.
- **inotify-tools** e uno script Bash per l'aggiornamento dinamico della blacklist.
- **Python3** e dipendenze da `requirements.txt` per eventuali script di enforcement policy.

Il Dockerfile esplicita questa configurazione, mentre l'avvio dei servizi è orchestrato tramite uno script `entrypoint.sh`.

Squid come proxy trasparente: Squid è configurato per funzionare come proxy trasparente. Ciò significa che il traffico HTTP dei client viene intercettato automaticamente senza necessità di configurare i client. Questa funzionalità è abilitata tramite `iptables`, che ridirige il traffico sulla porta 3128 di Squid (Listing 2).

Listing 2: Snippet di configurazione iptables per proxy trasparente

```
1 # REDIRECT (transparent proxy) - intercetta richieste al PEP (porta
      ↪ 3100)
2 iptables -t nat -A PREROUTING -s 172.20.0.2 -p tcp --dport 3100 -j
      ↪ REDIRECT --to-port 3128
3 iptables -t nat -A PREROUTING -s 172.21.0.2 -p tcp --dport 3100 -j
      ↪ REDIRECT --to-port 3128
4 iptables -t nat -A PREROUTING -s 172.22.0.2 -p tcp --dport 3100 -j
      ↪ REDIRECT --to-port 3128
```

I log di Squid sono accessibili all'interno della cartella `logs/squid` montata in un apposito volume, e vengono monitorati da Splunk.

Snort per l'analisi del traffico: Snort è configurato per analizzare il traffico in modalità promiscua su tutte le interfacce del container. Le regole personalizzate sono contenute nel file `local.rules` e permettono il rilevamento di scansioni di porte, attacchi DoS, Shell-code injection e tentativi di accesso anomali ai proxy. Nel Listing 3, un esempio di regola per rilevare uno scan SYN:

Listing 3: Regola Snort per SYN scan

```
1 alert tcp any any -> any any (flags: S; msg:"SYN scan attempt";
      ↪ threshold\:type threshold, track by\_src, count 5, seconds 10;
      ↪ sid:1000001; rev:1;)
```

I logs relativi agli alerts vengono salvati nel file `/var/log/snort/alert`, anch'esso monitorato da Splunk.

iptables - firewall e routing: Il file `rules.sh` contiene la configurazione completa di iptables. Le politiche predefinite sono restrittive (DROP), e solo specifici flussi vengono consentiti. Le regole stabiliscono il NAT per l'accesso a Internet, l'intercettazione per Squid, e la protezione delle porte sensibili (esempi in Listing 4).

Listing 4: Regole iptables semplificate

```
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT ACCEPT

# NAT per accesso a Internet
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE

# Consente traffico DNS e HTTP dei client interni
iptables -A FORWARD -s 172.28.0.0/16 -p udp --dport 53 -j ACCEPT
iptables -A FORWARD -s 172.28.0.0/16 -p tcp --dport 80 -j ACCEPT

# Blocca accesso diretto al database
iptables -A FORWARD -p tcp -d 172.24.0.3 --dport 5432 -j DROP
```

Meccanismo di blacklist dinamica: Il container include inoltre un sistema reattivo per il blocco dinamico di indirizzi IP considerati malevoli. Questo meccanismo si basa su un file testuale `/blacklist/blacklist.txt` che contiene un IP per riga. Un demone Bash[2] basato su `inotifywait` monitora il file e aggiorna automaticamente iptables (Listing 5):

Listing 5: apply_blacklist.sh - estratto

```
while inotifywait -e modify /blacklist/blacklist.txt; do
while read ip; do
iptables -C FORWARD -s "\$ip" -j DROP 2>/dev/null ||&#x20;
iptables -A FORWARD -s "\$ip" -j DROP
done < /blacklist/blacklist.txt
done
```

In questo modo, componenti esterni (es. il modulo Flask per l'enforcement delle policy) possono aggiungere IP al file per bloccarli in tempo reale.

Avvio dei servizi: Lo script `entrypoint.sh` avvia tutti i componenti in sequenza:

1. Avvia lo script `rules.sh` per impostare le regole iptables.
2. Esegue `apply_blacklist.sh` in background.
3. Avvia Snort in modalità log.
4. Avvia Squid come proxy.

[2]Script bash che viene eseguito in background come un demone, ovvero un processo che gira senta interazione dell'utente, che resta attivo in background e svolge attività ricorrenti o in attesa di eventi.

5. Mantiene il container in esecuzione con `tail -f`.

Questo approccio modulare consente un controllo centralizzato del traffico, con la possibilità di modificare e aggiornare dinamicamente le policy di sicurezza.

3.1.3 Policy Enforcement Point (PEP)

Il container `PEP` rappresenta un elemento cruciale dell'architettura Zero Trust, fungendo da intermediario tra i client e il database. Il suo compito principale è **intercettare e valutare ogni richiesta** di accesso ai documenti, applicare i meccanismi di autenticazione utente, e consultare il Policy Decision Point (PDP) per decidere se autorizzare o meno l'operazione richiesta.

Nel Dockerfile del container, basato sull'immagine `python:3.10-slim`, vengono installate solo le librerie indispensabili per mantenere il container leggero, tra cui `flask`, `requests`, `psycopg2-binary` (per la connessione a PostgreSQL), `bcrypt` e `cryptography` per la sicurezza e la criptazione dei file, oltre ad alcune dipendenze di sistema come `gcc` e `libpq-dev`.
Il container è configurato per operare su due reti Docker separate, `zt-gateway` e `zt-core`, per assicurare un corretto isolamento e una comunicazione controllata con gli altri componenti dell'architettura. Un volume è montato per garantire la persistenza e la sicurezza del file degli utenti (`users_db.json`).

Flusso di comunicazione: Implementato in Python e basato su Flask, il PEP riceve richieste HTTP dai clients (attraverso il gateway), per le operazioni di login, logout, o per richieste di accesso a documenti. Il processo seguito è il seguente:

1. **Autenticazione**: Tramite un endpoint dedicato (`POST/login`), il PEP autentica l'utente utilizzando hash sicuri (`bcrypt`), come mostrato nel Listing 6.

Listing 6: Autenticazione con bcrypt

```
1     def authenticate_user(username, password, user_db):
2     """
3     Autentica un utente verificando username e password.
4     Se l'autenticazione ha successo, restituisce True e il ruolo
      ↪ dell'utente.
5     Altrimenti, restituisce False e un messaggio di errore.
6     """
7     user = user_db.get(username)
8     if not user:
9         return False, "Utente non trovato"
10
11    if bcrypt.checkpw(password.encode(), user["password"].encode())
      ↪ :
12        return True, user["role"]
13    else:
14        return False, "Password errata"
15
```

2. **Sessione**: se l'autenticazione è valida, viene generato un token di sessione univoco, associato all'utente (Listing 7).

Listing 7: Generazione del token di sessione

```
    session.permanent = True # Sessione persistente
    session["username"] = username
    session["role"] = role_or_msg
    return jsonify({"status": "ok", "message": "Login riuscito"}),
    ↪ 200

```

3. **Gestione delle richieste**: `POST/request` è l'endpoint più importante, tramite cui il PEP riceve richieste di accesso o modifica a documenti. Dopo aver verificato che l'utente sia autenticato, il PEP costruisce una richiesta strutturata e la inoltra al PDP, includendo:

   - Identità dell'utente (username, ruolo);
   - Operazione richiesta (`read`, `write`, ecc.);
   - Tipo di documento e sensibilità;
   - Indirizzo IP del richiedente;
   - Timestamp della richiesta.

   Qualora al contrario la richiesta venga eseguita da un utente non autenticato, questa non viene permessa e, se necessario, il monitoraggio del traffico all'interno del gateway permette una corretta gestione di quest'ultima tramite l'applicazione delle apposite policy.

   Un estratto del codice di gestione è mostrato in Listing 8.

Listing 8: Inoltro della richiesta al PDP

```
    response = requests.post(PDP_URL, json={
    "timestamp": timestamp,
    "client_ip": client_ip,
    "username": username,
    "role": role,
    "operation": operation,
    "document_type": document_type
     }, timeout=20)

    decision = response.json().get("decision", "deny")
    if decision == "allow":
    # Esegue l'operazione (read/write)
    else:
    # Nega l'accesso

```

   In caso di risposta positiva del PDP (`allow`), il PEP esegue l'operazione richiesta. Queste operazioni sono incapsulate nei moduli contenuti nella cartella `db_scripts`, e gestite in modo trasparente dal PEP. La logica di accesso alle risorse è quindi separata da quella di autorizzazione, in accordo con i principi del modello Zero Trust.

4. **Logout**: una volta completata l'operazione, attraverso la rotta `POST/logout` viene eseguito il logout dell'utente.

Gestione sicura degli utenti Il modulo Python sviluppato per la gestione degli utenti (`user_auth.py`) implementa un sistema di autenticazione sicuro, progettato per essere semplice ma robusto. Questo sistema si basa su tre pilastri fondamentali: protezione delle password, cifratura dei dati, e separazione della chiave di sicurezza.

Per quanto riguarda la crittografia del database degli utenti, l'intero file `users_db.json` è cifrato utilizzando **Fernet**, un algoritmo simmetrico fornito dalla libreria `cryptography`. La chiave segreta utilizzata per la cifratura non è codificata nel codice sorgente, ma viene caricata da una variabile d'ambiente definita in un file `.env`, garantendo così una maggiore sicurezza.

Inoltre, le password non sono mai salvate in chiaro, vengono invece **hashate** utilizzando `bcrypt`, un algoritmo specificamente progettato per proteggere le credenziali. L'hash risultante è salvato nel database cifrato e viene confrontato al momento dell'autenticazione tramite `bcrypt.checkpw`.

Il modulo implementa le seguenti funzioni:

- `load_user_db()`: carica e decifra il file utenti. In caso di errore, restituisce un dizionario vuoto.
- `save_user_db(user_db)`: salva il database cifrandolo interamente.
- `create_user(username, password, role, user_db)`: crea un nuovo utente se non esiste già, cifrando la password e salvando il ruolo.
- `authenticate_user(username, password, user_db)`: verifica l'identità dell'utente confrontando l'hash della password.

Un esempio tipico di utilizzo prevede il caricamento del database, la creazione di un utente, e successiva autenticazione (Listing 9):

Listing 9: Esempio di utilizzo

```
1 user_db = load_user_db()
2 create_user("alice", "mypassword123", "admin", user_db)
3 authenticate_user("alice", "mypassword123", user_db)
```

Questo approccio consente di gestire in modo sicuro le credenziali utente in ambienti containerizzati o applicazioni web, mantenendo elevato il livello di protezione dei dati sensibili.

Interazione con il database: Entrando più in dettaglio nella gestione dell'interazione con il database documentale, per la gestione dell'accesso e delle operazioni, si è adottata una struttura modulare, ispirata ai principi del pattern *Data Access Object (DAO)*. I moduli che si occupano di questa interazione sono contenuti nella cartella `db_scripts` e comprendono tre file: `db_operations.py`, `db_DAO.py` e `db_exec.py`.
Il modulo `db_operations.py` implementa la classe `DatabaseManager`, che funge da gestore centrale della connessione al database PostgreSQL. Tale classe è strutturata secondo il pattern *Singleton*, garantendo così che venga mantenuta un'unica istanza attiva della connessione lungo tutto il ciclo di vita dell'applicazione. I parametri di connessione (host, nome del database, credenziali, ecc.) vengono recuperati dinamicamente tramite

variabili d'ambiente, rendendo il sistema flessibile e sicuro anche in fase di deploy.
Il modulo `db_DAO.py` definisce invece la classe `FileDocumentoDAO`, incaricata di incapsulare tutte le operazioni CRUD (*Create, Read, Update, Delete*) sulla tabella `file_documenti`. Tale classe utilizza l'istanza del `DatabaseManager` per eseguire query e aggiornamenti in maniera centralizzata e riutilizzabile. Le operazioni supportate comprendono:

- l'inserimento di nuovi documenti con nome, contenuto e livello di sensibilità;
- la lettura di un documento a partire dal suo ID;
- l'aggiornamento selettivo dei campi di un documento esistente;
- la cancellazione di un documento dato il suo ID.

Infine, il modulo `db_exec.py` funge da livello intermedio tra il backend dell'applicazione (ad esempio il PEP o i microservizi) e il DAO. Fornisce funzioni di alto livello per eseguire operazioni di scrittura e lettura, applicando anche controlli di autorizzazione sul ruolo dell'utente. In particolare, per l'operazione di lettura viene impedito l'accesso a documenti etichettati come `sensibile` a utenti che non abbiano il ruolo di `Direttore`, secondo le politiche definite.
Questa organizzazione modulare, oltre a garantire separazione delle responsabilità e riusabilità del codice, consente una facile manutenibilità e scalabilità futura del sistema.

3.1.4 Policy Decision Point (PDP)

Il container `pdp` (Policy Decision Point) rappresenta il cuore del processo decisionale all'interno dell'architettura Zero Trust. Il suo compito è valutare, sulla base di una serie di criteri di sicurezza e contesto, se un'operazione richiesta da un utente debba essere permessa o negata. Attraverso un approccio modulare e policy-driven, riesce infatti a valutare dinamicamente richieste di accesso e comportamenti di rete, interagendo con altri componenti (Splunk, PEP) e contribuendo a rafforzare la sicurezza dell'intera architettura. In dettaglio, il servizio, sviluppato in Python e basato su Flask, espone due endpoint fondamentali:

- `/update_trust`: riceve eventi da Splunk relativi alla sicurezza della rete e aggiorna di conseguenza il punteggio di fiducia degli IP coinvolti, eventualmente bloccandoli.
- `/decide`: riceve richieste dal PEP (Policy Enforcement Point) e valuta se un'operazione (es. lettura, scrittura, cancellazione di documenti) sia ammissibile, in base al ruolo dell'utente, alla fiducia della rete e al tipo di documento.

Il `Dockerfile` è costruito a partire dall'immagine ufficiale `python:3.10-slim`. All'interno del container viene impostata la directory di lavoro `/app`, e in essa vengono copiati tutti i file essenziali all'esecuzione del servizio, tra cui:

- `pdp.py`: script principale che espone gli endpoint e implementa la logica di decisione.
- `policies.py`: contiene le politiche di valutazione della rete.
- `utils.py`: include funzioni di utilità (es. accesso a file, gestione della blacklist, aggiornamento punteggi).
- `trust_db.json`: database dei punteggi di fiducia.

- `GeoLite2-Country.mmdb`: database geografico per associare IP a stati.
- `entrypoint.sh`: script di avvio che resetta la blacklist ad ogni avvio.

Le dipendenze vengono installate tramite `pip` (`Flask`, `dotenv`, `cryptography`, `geoip2`, ecc.). Infine, viene impostato l'entrypoint e il comando di default `python pdp.py`.

Trust Updating: L'endpoint `/update_trust` riceve notifiche di eventi anomali da Splunk attraverso gli webhooks inviati attraverso le saved searches (definite in seguito), filtrandole e indirizzandole al corrispondente meccanismo di gestione per aggiornare dinamicamente la fiducia associata a determinati indirizzi IP. Le policy gestite in questo caso includono:

- **TrustReputation-Increase**: incrementa la fiducia in seguito a comportamento corretto.
- **TrustReputation-Decrease**: riduce la fiducia, ad esempio in caso di DoS o altre minacce.
- **Snort-Attack-Detection-30Days**: blocca direttamente gli IP coinvolti in attacchi rilevati da Snort.
- **PortScanning-HighRate-Detection**, **ShellCode-Injection-Detection**: causano il blocco dell'IP.
- **Non-Working-Hours-Detection-More-Than-10-IPs**: penalizza IP attivi in orari anomali.

In questo modo, la struttura modulare consente un'agevole estensione futura delle politiche.

Decision: L'endpoint `/decide` è responsabile della valutazione completa di una richiesta di accesso ad una risorsa. Viene infatti chiamato dal Policy Enforcement Point per determinare se un'operazione richiesta da un utente debba essere consentita o negata, sulla base di una serie di condizioni. Più in dettaglio, l'esito si basa sui seguenti controlli:

1. **Verifica dell'IP**: se è nella `blacklist.txt`, l'accesso è negato immediatamente.
2. **Valutazioni di rete**: nel `policies.py`, sono definite delle funzioni utilizzate per effettuare controlli sulla provenienza IP, e sulla localizzazione geografica, abbassando o aumentando il punteggio di fiducia di conseguenza.
3. **Calcolo della fiducia**: Un punteggio combinato viene ottenuto come media tra la fiducia di rete e quella associata al ruolo dell'utente.
4. **Confronto con soglie**: ogni tipo di operazione (`read`, `write`, `delete`) su un certo tipo di documento (`Dati Personali`, `Dati Transazionali`, ecc.) ha una soglia minima richiesta di fiducia (Tabella 1).
5. **Controllo permessi del ruolo**: Con l'ausilio di funzioni definite nei moduli `policies.py` e `utils.py`, si verifica che l'operazione e il tipo di documento richiesti siano compatibili col ruolo.

Tali controlli rendono il processo decisionale contestuale, dinamico e coerente con il paradigma Zero Trust.
L'esito dei controlli viene poi inviato al Policy Enforcement Point. Se tutte le condizioni sono soddisfatte, la decisione è `allow` e l'operazione può essere autorizzata, altrimenti è `deny`.

Sicurezza e resilienza: Il file `trust_db.json`, che contiene il punteggio di fiducia degli indirizzi IP, viene cifrato e decifrato utilizzando l'algoritmo simmetrico `Fernet` fornito dalla libreria `cryptography`. La chiave di cifratura, caricata tramite variabile d'ambiente (`TRUST_KEY`), non è scritta nel codice, ma viene gestita in modo sicuro attraverso un file `.env`.

Ogni volta che il database viene letto o aggiornato, i dati vengono:

- **Decifrati al momento della lettura** tramite la funzione `load_trust_db()`, che recupera il contenuto binario del file, lo decifra con `fernet.decrypt()`, e lo deserializza in un oggetto Python con `json.loads()`.
- **Cifrati prima della scrittura** mediante la funzione `save_trust_db()`, che serializza il dizionario Python in una stringa JSON, la codifica in byte e la cifra con `fernet.encrypt()`, scrivendo infine il risultato cifrato su disco.

Questo approccio garantisce che nessun dato sensibile (come gli indirizzi IP con i rispettivi punteggi) venga mai memorizzato in chiaro, aumentando la resilienza del sistema contro accessi non autorizzati o compromissioni del contenitore.
Inoltre, in caso di punteggi di fiducia troppo bassi, l'IP viene automaticamente aggiunto alla blacklist persistente, rendendo il sistema capace di reagire in tempo reale a comportamenti anomali o malevoli.

3.1.5 Database Container

Il container `db` ospita un'istanza di **PostgreSQL**, la risorsa informativa protetta all'interno dell'architettura Zero Trust. Questo database conserva documenti e dati di vario livello di sensibilità, il cui accesso è rigidamente controllato tramite autorizzazioni fornite dal modulo PEP, che a sua volta agisce solo in seguito a una decisione positiva del PDP.

All'avvio, il database viene inizializzato automaticamente grazie alla presenza di uno o più file montati nel volume `/docker-entrypoint-initdb.d/` (Listing 11). L'immagine ufficiale di PostgreSQL esegue automaticamente tutti i file SQL e gli script Bash presenti in quella directory, purché il volume dati (`db_data`) sia inizialmente vuoto. Questo consente la creazione della struttura dati e la configurazione del database senza intervento manuale.

In particolare:

- `schema.sql` definisce la struttura della tabella `file_documenti`, contenente i file accessibili nel sistema e la loro classificazione secondo una sensibilità dichiarata tramite un tipo `ENUM` (sensibile / non_sensibile);
- `data.sql` popola la tabella con un insieme di file esemplificativi, distinti per contenuto e livello di riservatezza (Listing 10).

Listing 10: esempi contenuti in data.sql

```
-- File/documenti di esempio
INSERT INTO file_documenti (nome_file, contenuto, sensibilita)
    ↪ VALUES
('Dati Personali Cliente1', 'Nome: Mario Rossi\nCF:
    ↪ CLNTCF01A01H501Z\n...', 'sensibile'),
('Saldo Cliente1', 'Saldo: 10000 EUR', 'sensibile'),
('Ricevuta Bonifico Cliente1', 'Bonifico in entrata: 500 EUR', '
    ↪ sensibile'),
('Elenco Clienti', 'Cliente1, Cliente2, ...', 'non_sensibile'),
```

Listing 11: Definizione del container db in docker-compose.yml

```
db:
  image: postgres:latest
  container_name: db
  environment:
    PGDATA: /var/lib/postgresql/data/pgdata
    POSTGRES_USER: ${POSTGRES_USER}
    POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
    POSTGRES_DB: ${POSTGRES_DB}
  networks:
    zt-core:
      ipv4_address: 172.25.0.4
  ports:
    - "5432:5432"
  volumes:
    - ./db/schema.sql:/docker-entrypoint-initdb.d/01_schema.sql
    - ./db/data.sql:/docker-entrypoint-initdb.d/02_data.sql
    - ./db/certs:/certs:ro
    - ./db/init/03_ssl.sh:/docker-entrypoint-initdb.d/03_ssl.sh:ro
    - db_data:/var/lib/postgresql/data
```

L'accesso ai documenti archiviati avviene esclusivamente tramite interrogazioni controllate, condizionate dalle policy valutate dal PDP in base al contesto di richiesta, alle credenziali utente e alla sensibilità del file.

Comunicazione sicura con TLS Per aderire ai principi del modello Zero Trust, in particolare quelli relativi a confidenzialità, integrità e autenticazione, il canale di comunicazione tra il PEP e il database PostgreSQL è protetto da *TLS (Transport Layer Security)*. Questo assicura che il client (PEP) comunichi solo con il vero server autorizzato (autenticazione), i dati in transito siano cifrati (riservatezza), ed eventuali alterazioni vengano rilevate (integrità).
PostgreSQL offre supporto nativo a TLS, il che ha permesso una configurazione relativamente semplice del canale cifrato. I certificati necessari per la cifratura e l'identificazione sono generati tramite comandi da terminale e salvati all'interno della directory `./db/certs`. Nello specifico:

- `server.crt`: certificato del server utilizzato per autenticare il database agli occhi del client (PEP).

- `server.key`: chiave privata del database, utilizzata per la cifratura della comunicazione.

Durante l'inizializzazione del container, lo script `03_ssl.sh` automatizza la configurazione del server PostgreSQL affinché accetti solo connessioni TLS. Questo approccio, oltre a essere compatibile con ambienti complessi (come Windows, dove i permessi sui file possono causare problemi), garantisce la ripetibilità e la corretta applicazione della configurazione a ogni avvio. In Listing 12, un esempio di configurazione.

Listing 12: esempio di configurazione automatica tramite script

```
1 echo "ssl = on"
2 echo "ssl_cert_file = '/var/lib/postgresql/server.crt'"
3 echo "ssl_key_file  = '/var/lib/postgresql/server.key'"
```

Le operazioni svolte all'interno del file bash sono le seguenti:

- **Copia dei certificati**: i file `server.crt` e `server.key` vengono copiati dalla directory `/certs` (montata come volume) alla directory dati di PostgreSQL (`/var/lib/postgresql`).
- **Impostazione dei permessi**: il file della chiave privata viene protetto con permessi restrittivi (`chmod 600`), mentre il certificato pubblico riceve permessi di lettura più permissivi (`chmod 644`). Entrambi i file sono assegnati all'utente `postgres`.
- **Modifica del file di configurazione** `postgresql.conf`: vengono aggiunte le direttive per abilitare TLS e specificare i percorsi dei certificati.
- **Aggiornamento di** `pg_hba.conf`: se non già presente, viene aggiunta una riga che abilita connessioni client TLS da qualsiasi IP (`0.0.0.0/0`) utilizzando il metodo di autenticazione `md5`.

Grazie al TLS, la comunicazione tra PEP e database è protetta da intercettazioni (confidenzialità), manipolazioni (integrità) e impersonificazioni (autenticazione). Il PEP, prima di ogni interazione col database, verifica l'identità del server attraverso il certificato e instaura una connessione sicura, rendendo il sistema conforme ai principi fondamentali del modello Zero Trust.

3.1.6 Monitoraggio e logging centralizzato: container Splunk

Il container `splunk` ha il compito di fornire un sistema di **monitoraggio centralizzato dei log**, elemento essenziale all'interno dell'architettura Zero Trust. In particolare, Splunk riceve e indicizza i file di log provenienti da **Squid**, il proxy descritto in Sezione 2.2, utile per registrare tutte le richieste HTTP effettuate dai client, e **Snort**, il sistema di intrusion detection e prevenzione, descritto in Sezione 2.3, che produce alert in tempo reale su attività sospette o malevole.
Splunk consente di analizzare, filtrare e visualizzare tali log tramite dashboard interattive e ricerche personalizzate, fornendo così una visione chiara e in tempo reale dello stato della rete e del comportamento degli utenti.

Ruolo nel modello Zero Trust Nel contesto della filosofia Zero Trust, Splunk agisce come componente chiave del **monitoraggio continuo**, uno dei pilastri fondamentali del paradigma. Il sistema non si limita alla semplice raccolta di log, ma:

- individua accessi anomali, comportamenti sospetti e violazioni delle policy;
- genera **alert automatici** in presenza di eventi critici (es. attacco brute force, accesso a siti bloccati);
- trasmette tali alert al Policy Decision Point (Sezione 3.1.4) tramite *webhook HTTP*, abilitando l'enforcement dinamico delle contromisure di sicurezza.

Configurazione del container Splunk Nel file `docker-compose.yml`, il container `splunk` viene configurato utilizzando l'immagine ufficiale `splunk/splunk:latest` (Listing 13), accettando automaticamente la licenza all'avvio e impostando la password dell'utente amministratore tramite la variabile d'ambiente `SPLUNK_PASSWORD`. Il container è connesso alla rete `zt-core` con indirizzo IP statico `172.25.0.5`, così da risultare facilmente raggiungibile dagli altri container del sistema.

Listing 13: Definizione del container Splunk

```
splunk:
  image: splunk/splunk:latest
  container_name: splunk
  environment:
    - SPLUNK_START_ARGS=--accept-license
    - SPLUNK_PASSWORD=${SPLUNK_PASSWORD}
  ports:
    - "8000:8000"     # Interfaccia web
    - "8088:8088"     # HEC endpoint per webhook
    - "8089:8089"     # API di gestione
  networks:
    zt-core:
      ipv4_address: 172.25.0.5
  volumes:
    - ./logs/squid:/var/log/squid
    - ./logs/snort:/var/log/snort
    - ./splunk-apps/log_inputs:/opt/splunk/etc/apps/log_inputs
    - ./splunk-apps/dashboard:/opt/splunk/etc/apps/dashboard
    - ./splunk-apps/lookups:/opt/splunk/etc/apps/search/lookups
    - splunk_etc:/opt/splunk/etc
    - splunk_var:/opt/splunk/var
```

Le porte esposte dal container sono fondamentali per l'interazione con Splunk: la porta `8000` consente l'accesso all'interfaccia web, mentre la `8088` è riservata all'endpoint HEC (HTTP Event Collector), che riceve i webhook generati in seguito ad alert automatici. La porta `8089`, infine, è utilizzata per le API interne di gestione e orchestrazione.

Sul container vengono inoltre montati una serie di volumi per garantire la persistenza dei dati e l'integrazione con il resto del sistema. In particolare, i log generati dai container `squid` e `snort` vengono montati rispettivamente nei percorsi `/var/log/squid` e `/var/log/snort`, rendendoli immediatamente disponibili a Splunk per l'indicizzazione. Inoltre, vengono montate tre app personalizzate, sviluppate appositamente per questo

progetto: `log_inputs`, che definisce gli input da file locali, `dashboard`, che fornisce visualizzazioni grafiche e interattive, e una directory contenente i file di *lookup*, utili per arricchire i dati durante le ricerche. I volumi `splunk_etc` e `splunk_var` assicurano infine la persistenza della configurazione e dei dati indicizzati, anche in caso di riavvio del container.

3.1.7 Applicazioni Personalizzate per la Piattaforma Splunk

Splunk consente l'estensione delle sue funzionalità tramite il concetto di *applicazioni*, ovvero pacchetti modulari che possono includere configurazioni di input, dashboard, ricerche salvate, azioni di alert, lookup e script personalizzati. Queste app possono essere sviluppate internamente e installate nella piattaforma per adattarla a casi d'uso specifici.
Nel contesto del presente progetto, l'utilizzo di app customizzate si è rivelato essenziale per strutturare in maniera modulare e manutenibile le componenti della pipeline di monitoraggio e sicurezza. Le applicazioni personalizzate permettono infatti di:

- configurare in modo chiaro e riutilizzabile le fonti di dati, come i log generati da Squid e Snort;
- progettare dashboard interattive per la visualizzazione e l'analisi dei dati raccolti;
- definire azioni automatizzate in risposta ad eventi di sicurezza rilevati, come l'invio di webhook a un modulo di enforcement.

Le app sviluppate sono collocate nel container Splunk all'interno della directory `/opt/splunk/etc/apps`, e vengono caricate automaticamente all'avvio del sistema. L'approccio modulare facilita il versionamento, il debug e l'evoluzione incrementale del sistema di monitoraggio, garantendo allo stesso tempo coerenza e isolamento.

log_inputs: Ingestione e Analisi Automatica dei Log L'applicazione `log_inputs` rappresenta uno dei componenti fondamentali dell'architettura Splunk impiegata in questo progetto. Il suo scopo principale è quello di configurare in modo modulare e riutilizzabile l'ingestione dei log prodotti dai principali strumenti di sicurezza e monitoraggio, ovvero **Squid** e **Snort**, e di attivare ricerche programmate (*saved searches*) che automatizzano l'analisi e la reazione ad eventi anomali o comportamenti malevoli rilevati nei log.

L'ingestione dei log è definita all'interno del file `local/inputs.conf` attraverso le direttive `[monitor://...]`, che indicano a Splunk quali file devono essere costantemente monitorati. I files in questione sono stati montati in un volume nel container `splunk` (si faccia riferimento alla sezione 3.1.6).
Ogni direttiva specifica il percorso assoluto del file da monitorare, l'indice in cui i dati devono essere archiviati (in questo caso `main`) e il tipo di sorgente (`sourcetype`) da associare, che aiuta Splunk nella normalizzazione e parsing degli eventi. Un esempio di configurazione è il seguente (Listing 14):

Listing 14: Monitoraggio dei log generati da Squid e Snort

```
1 [monitor:///var/log/squid/access.log]
2 sourcetype = squid
3 index = main
4
```

```
[monitor:///var/log/snort/alert]
sourcetype = snort_alert
index = main
```

In questo modo, ogni nuova riga scritta da Squid o Snort in uno dei file monitorati viene immediatamente indicizzata da Splunk e resa disponibile per interrogazioni in tempo reale, visualizzazioni o correlazioni con altri eventi.

Per quanto riguarda le azioni di alert, il file `local/savedsearches.conf` contiene la definizione di una serie di *saved searches* (esempio in Listing 15), ovvero ricerche pianificate che vengono eseguite automaticamente con frequenza configurabile, ad esempio ogni minuto o ogni due minuti nel contesto di test. Tali ricerche implementano in background un monitoraggio continuo del traffico analizzando i logs monitorati in determinati periodi di tempo, permettendo la messa in atto delle politiche di sicurezza definite nel progetto e sono in grado di generare alert in caso di corrispondenza con condizioni sospette. Ogni alert è associato a un *webhook* che invia una richiesta HTTP POST all'endpoint `update_trust` del Policy Decision Point (vedi Sez. 3.1.6), il quale aggiorna la reputazione degli IP coinvolti e applica i meccanismi di gestione corrispondenti.

Listing 15: Esempio di saved search con webhook associato

```
[Snort-Attack-Detection-30Days]
search = | inputlookup known_clients.csv\
         | join src_ip\
         [ search index=main sourcetype=snort_alert earliest=-30Days
    ↪ latest=now\
              | rex field=_raw "(?<src_ip>\d{1,3}(?:\.\d{1,3}){3}):\d+\s
    ↪ +->"\
              | stats count by src_ip\
              | where count >= 60\
         ]\
         | table src_ip
dispatch.earliest_time = -30d@d
dispatch.latest_time = now
cron_schedule = */2 * * * *
enableSched = 1
action.webhook = 1
action.webhook.param.url = http://pdp:5050/update_trust
action.webhook.httpmethod = POST
```

Facendo riferimento al codice in Listing 15, i parametri principali utilizzati per la definizione delle *saved searches*, in particolare per la gestione dell'intervallo temporale e della schedulazione, sono i seguenti:

- `dispatch.earliest_time` e `dispatch.latest_time`: definiscono rispettivamente l'inizio e la fine dell'intervallo temporale da analizzare nei log. Nel contesto del progetto, l'intervallo è generalmente impostato sugli ultimi 30 giorni (`-30d@d` fino a `now`) per garantire una visione completa e significativa dello storico.
- `cron_schedule`: specifica la frequenza con cui la ricerca viene eseguita. Ad esempio, `*/2 * * * *` corrisponde a un'esecuzione ogni 2 minuti. Questa configurazione è utile per ambienti di test ad alta frequenza, mentre in produzione le ricerche potrebbero essere schedulate con intervalli più lunghi (ad esempio mensili per politiche di reputazione positiva).

- `enableSched`: attiva la schedulazione automatica della ricerca (`1` per abilitata).
- `action.webhook.*`: gruppo di parametri che definisce l'azione associata all'alert, ovvero l'invio di una richiesta HTTP POST a un endpoint remoto per l'aggiornamento della fiducia dell'IP coinvolto. Tra questi:
  - `action.webhook.param.url`: URL di destinazione del webhook.
  - `action.webhook.httpmethod`: metodo HTTP da utilizzare, tipicamente `POST`.

In dettaglio, le altre *saved searches* implementate sono le seguenti:

- `Non-Working-Hours-Detection`: penalizza gli IP che, nell'arco di 30 giorni, hanno effettuato un numero significativo di richieste (superiori a una soglia) fuori dall'orario lavorativo, definito tra le 08:00 e le 20:00. Questo consente di rilevare comportamenti anomali o potenzialmente sospetti, come accessi notturni.
- `TrustReputation-Increase`: premia i client che, negli ultimi 30 giorni, non sono stati coinvolti in eventi di sicurezza rilevati da Snort. In un contesto reale, questa ricerca verrebbe eseguita con cadenza mensile, ma è stata temporaneamente schedulata con frequenza più alta per scopi di test.
- `PortScanning-HighRate-Detection` e `ShellCode-Injection-Detection`: rilevano comportamenti malevoli specifici. La prima individua attacchi di port scanning ad alta intensità (almeno 70 tentativi in un solo minuto) verso la rete interna, mentre la seconda rileva tentativi di injection di shellcode a danno del sistema.
- `Snort-Attack-Detection-30Days`: monitora gli IP che, nell'arco di 30 giorni, hanno generato un numero elevato di allarmi Snort (almeno 60), indicativo di comportamenti ostili ripetuti.
- `TrustReputation-Decrease`: riduce la fiducia per tutti gli IP che hanno condotto attacchi riconducibili a tentativi di Denial of Service (DoS) sulla risorsa PEP, secondo quanto rilevato da Snort nell'ultimo mese.

Ogni ricerca impiega operatori SPL (*Search Processing Language*) avanzati, come `rex`, `stats`, `join`, e `eval`, per eseguire parsing dei log, correlazioni tra eventi e trasformazioni temporali, garantendo un alto livello di automazione e precisione.

Dashboard: Integrazione e Visualizzazione dei Dati L'integrazione con Splunk ha permesso di realizzare dunque un potente sistema di monitoraggio centralizzato, in grado di raccogliere, analizzare e visualizzare in tempo reale i dati provenienti dai componenti critici dell'architettura Zero Trust. Sono state allora sviluppate due dashboard, concepite per fornire una visualizzazione diretta e continua sul traffico di rete e sugli eventi di sicurezza. Le dashboard sono state anch'esse realizzate tramite una applicazione Splunk personalizzata, `dashboard`.
Ogni dashboard è definita tramite un file XML che struttura i pannelli (panels), i moduli di ricerca (search modules), le visualizzazioni (visualizations) e i campi interattivi (form inputs), sfruttando il linguaggio *Simple XML* fornito da Splunk.
Questo approccio garantisce una maggiore flessibilità nella personalizzazione della visualizzazione e nella composizione della dashboard, e consente anche una facile esportazione o versionamento tramite sistemi di controllo del codice sorgente.

Più in dettaglio, le dashboard sviluppate forniscono un'interfaccia intuitiva per il monitoraggio e l'analisi dei log generati dai componenti chiave del sistema: il proxy Squid e il sistema di intrusion detection Snort. Entrambe sono accessibili tramite l'interfaccia web di Splunk e aggiornano i dati in tempo reale grazie all'esecuzione automatica delle query SPL (*Search Processing Language*).

La **Squid Traffic Monitoring Dashboard** è progettata per monitorare e analizzare il traffico web che transita attraverso il proxy Squid. I log generati da Squid vengono raccolti automaticamente da Splunk, indicizzati e interrogati tramite ricerche SPL per fornire una rappresentazione visuale dettagliata del comportamento di navigazione degli utenti.

Le principali componenti della dashboard includono:

- **Trend temporale delle richieste**: un grafico ad area mostra l'andamento orario delle richieste HTTP/HTTPS processate dal proxy negli ultimi 30 giorni (Figura 8). Questa vista consente di individuare picchi anomali o fluttuazioni del traffico.

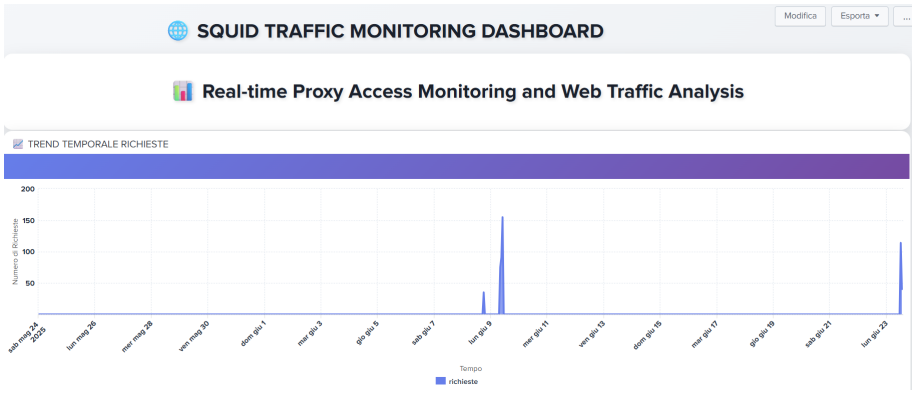

Figura 8: Trend temporale delle richieste

- **Top 10 URL più richiesti**: un pannello classifica i siti maggiormente visitati, con visualizzazione a barre della percentuale di traffico generato da ciascun dominio (Figura 9).
- **Top 10 IP client più attivi**: una tabella dinamica elenca gli indirizzi IP dei client che effettuano il maggior numero di richieste, utile per identificare comportamenti sospetti o uso eccessivo della banda (Figura 9).
- **Statistiche riassuntive**: indicatori numerici evidenziano il numero totale di richieste e di URL unici visitati nelle ultime 24 ore, fornendo una sintesi immediata del volume e della varietà del traffico (Figura 10).
- **Distribuzione dei metodi HTTP**: un grafico a torta mostra la frequenza relativa dei metodi HTTP (GET, POST, CONNECT, ecc.), utile per identificare tentativi di tunneling o utilizzi anomali del proxy (Figura 10).

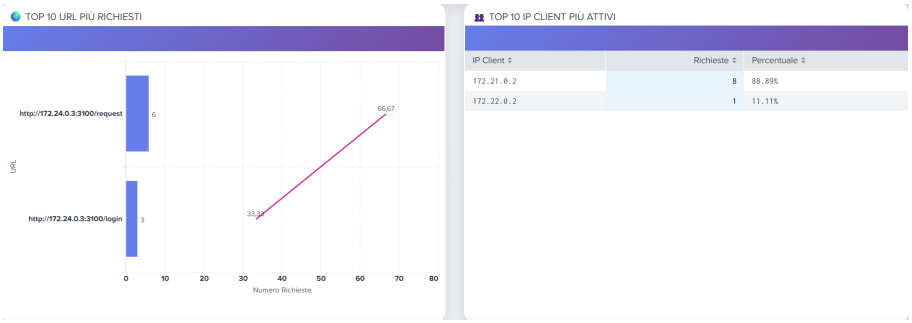

Figura 9: Top 10 URL più richiesti e Top 10 client più attivi

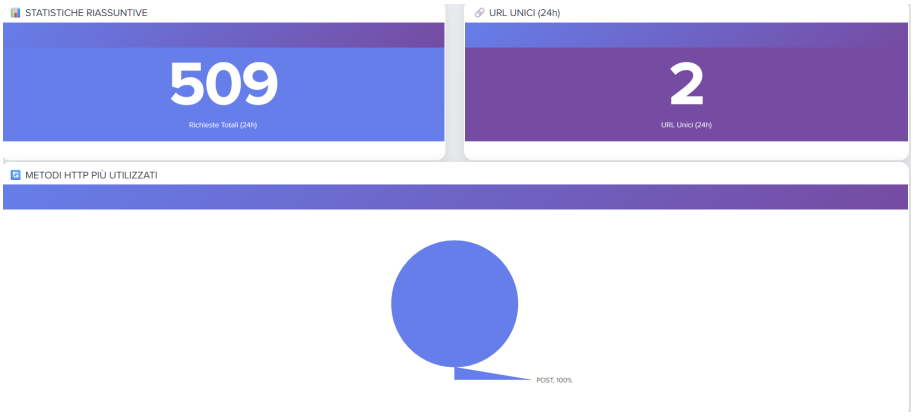

Figura 10: Statistiche Riassuntive e Distribuzione dei metodi HTTP

La **Snort Attack Dashboard** (Figura 11) si focalizza invece sul rilevamento e l'analisi degli eventi di sicurezza generati da Snort, il sistema di rilevamento delle intrusioni configurato sul gateway. Gli alert generati da Snort vengono inviati in tempo reale a Splunk, dove vengono elaborati e visualizzati per supportare l'analisi forense e la risposta agli incidenti.

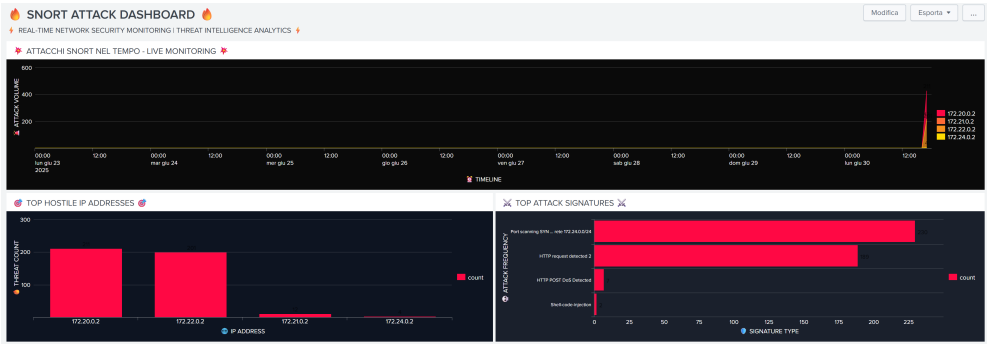

Figura 11: Dashboard Snort

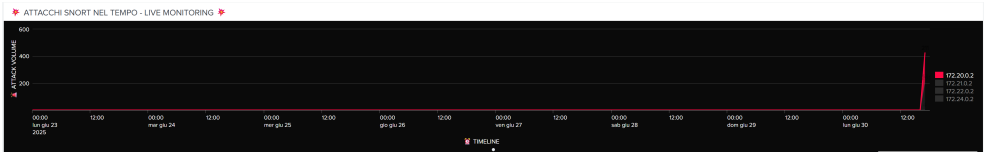

Figura 12: Zoom timeline dashboard Snort con filtro applicato per IP 172.20.0.2

Figura 13: Zoom timeline dashboard Snort con filtro applicato per IP 172.22.0.2

Le funzionalità principali della dashboard includono:

- **Trend temporale degli attacchi**: un grafico ad area mostra la distribuzione oraria degli attacchi rilevati, raggruppati per indirizzo IP sorgente. Questa vista permette di evidenziare picchi di attività malevola e pattern ricorrenti.
- **Top 10 IP ostili**: un elenco dei dieci indirizzi IP che hanno generato il maggior numero di alert negli ultimi due giorni, facilitando l'identificazione di sorgenti persistenti di attacco.
- **Top 10 signature di attacco**: una classifica delle signature più frequentemente attivate da Snort, che fornisce informazioni sulla natura delle minacce prevalenti (es. scansioni di porta, attacchi DoS, tentativi di exploit noti).

Grazie all'uso combinato di SPL, XML e funzionalità grafiche avanzate, queste dashboard costituiscono un'interfaccia analitica fondamentale per la gestione della sicurezza e la visibilità operativa all'interno del sistema Zero Trust implementato.

4 Policy implementate

All'interno della nostra architettura Zero Trust, l'enforcement delle policy di sicurezza rappresenta dunque il cuore del sistema di difesa proattivo e adattivo. Come descritto nei capitoli precedenti, questo è stato implementato tramite il monitoraggio centralizzato con Splunk, il rilevamento delle minacce in tempo reale con Snort, il controllo delle richieste tramite proxy (Squid), l'applicazione delle regole iptables e attraverso la dinamica di gestione del trust score effettuata dal PDP.
Le decisioni di autorizzazione vengono prese in base a una valutazione congiunta di parametri comportamentali (es. orari di accesso, frequenza di attacchi), contestuali (rete di provenienza, geolocalizzazione), e statici (ruolo utente, operazione richiesta, tipo di risorsa). Il sistema è stato progettato per essere modulare e facilmente estendibile, in modo da poter adattare le regole di sicurezza a contesti differenti o a esigenze future.
In particolare, le policy sono state codificate sotto forma di regole attivate sia in modo reattivo (a seguito di eventi specifici come attacchi rilevati o accessi anomali) sia in modo proattivo (tramite valutazioni periodiche dello stato di fiducia degli IP e degli utenti). Ogni policy è associata a una specifica azione, come l'aggiornamento del punteggio di fiducia, il blocco della comunicazione, o la restrizione di accessi.

Le policy attualmente attive nel sistema sono dunque le seguenti:

1. **Reputazione storica degli IP**: la fiducia verso un IP viene modificata sulla base del suo comportamento nel tempo:
   - Se un IP ha generato più di 60 alert Snort negli ultimi 30 giorni, viene bloccato attraverso l'inserimento immediato nella blacklist (`Snort-Attack-Detection-30Days`).
   - Se si verificano più di 5 accessi fuori orario lavorativo (20:00–08:00) nell'arco di 30 giorni, il punteggio di fiducia viene ridotto di 15 punti (`Non-Working-Hours-Detection-More-Than-10-IPs`)
   - In caso di comportamenti sospetti, come HTTP POST anomali simili a DoS verso il PEP, si ha una penalità di 40 punti nel punteggio di fiducia (`TrustReputation-Decrease`)
   - Se un IP non ha generato eventi malevoli per 30 giorni, viene assegnato un bonus di 1 punto nel punteggio di fiducia (`TrustReputation-Increase`.
2. **Monitoraggio in tempo reale degli attacchi**:
   - Le attività di **Port Scanning** comportano un blocco immediato dell'ip attraverso l'inserimento di quest'ultimo nella blacklist (`PortScanning-HighRate-Detection`)
   - I tentativi di **Shellcode Injection** rilevati nei log di Snort comportano un blocco immediato dell'ip attraverso l'inserimento di quest'ultimo nella blacklist (`ShellCode-Injection-Detection`)
3. **Controllo in tempo reale della rete di origine**: L'affidabilità, o punteggio di fiducia, viene aggiornata dinamicamente in base alla subnet da cui proviene la richiesta:
   - `Check rete interna (172.20.0.0/16)` → **+10 punti fiducia**.

- `Check rete esterna (172.21.0.0/24)` → **-5 punti fiducia**.
- `Check Wi-Fi (172.22.0.0/16)` → **-5 punti fiducia**.

4. **Geolocalizzazione**: La localizzazione dell'IP influenza il trust score:
   - Se l'IP è localizzato fuori dall'Italia, viene imposta una penalità di 40 punti nel punteggio di fiducia (`Check IP location`)
5. **Controllo orario delle richieste**: Le richieste effettuate al di fuori dell'orario lavorativo sono soggette a verifica tramite proxy (Squid), e conseguente penalizzazione penalizzazione (`Non-Working-Hours-Detection-More-Than-10-IPs`).
6. **Autenticazione utente e fiducia basata sul ruolo**:
   - Nessuna operazione è consentita senza autenticazione dell'utente.
   - Ogni ruolo dispone di un punteggio di fiducia iniziale:
     - Direttore: 85
     - Consulente: 75
     - Cassiere: 70
     - Cliente: 60
7. **Autorizzazione alle operazioni sulle risorse**: Per autorizzare un'operazione su una risorsa, devono essere soddisfatte contemporaneamente le seguenti condizioni:
   - Il ruolo dell'utente consente l'operazione richiesta (Tabella 2).
   - Il ruolo ha accesso alla risorsa richiesta (Tabella 2).
   - La media tra punteggio di fiducia dell'IP e dell'utente supera la soglia richiesta (Tabella 1).

Tipo Documento	**Operazione**	**Soglia minima**
Dati personali	read / write / delete	60 / 80 / 80
Dati transazionali	read / write / delete	65 / 75 / 80
Documenti operativi	read / write / delete	60 / 70 / 80

Tabella 1: Soglie minime richieste per eseguire operazioni sui documenti

Ruolo	**Permessi**
Direttore	Tutte le operazioni su tutti i documenti
Consulente	Solo lettura e scrittura su documenti operativi
Cassiere	Lettura e scrittura su documenti transazionali e operativi
Cliente	Sola lettura su documenti personali

Tabella 2: Permessi basati sul ruolo

5 Testing

Per validare il corretto funzionamento dell'infrastruttura Zero Trust e delle policy di sicurezza implementate, sono stati realizzati degli script interattivi di test (`uc.sh`) collocati all'interno di ciascun container client: `client_internal`, `client_external` e `client_wifi`.
Tali script permettono di simulare sia l'utilizzo regolare del sistema (operazioni di accesso e gestione documentale) sia scenari di attacco leciti, al fine di valutare l'efficacia dei meccanismi di controllo e rilevamento delle minacce (es. Snort, blocchi dinamici via PEP/PDP).
Questa fase di test è cruciale per garantire che l'intero ciclo di rilevamento, valutazione e risposta agli eventi avversi sia operativo e rispetti i requisiti dell'architettura Zero Trust implementata.

5.1 Esecuzione degli script

Per eseguire lo script di test all'interno di un container, è sufficiente utilizzare il comando `docker exec` per accedere al terminale del container desiderato ed eseguire lo script. Di seguito un esempio per avviare lo script nel client interno:

Esecuzione dello script di test

```
docker exec -it client_internal bash ./uc.sh
```

5.2 Funzionalità dello script `uc.sh`

Lo script presenta un menu interattivo che consente di accedere alle seguenti funzionalità:

- **Login/Logout**: permette di autenticarsi con credenziali di utenti registrati e gestire la sessione mediante cookie HTTP.
- **Operazioni sui documenti**:
  - `read` – Lettura di un documento, previa indicazione dell'ID, della tipologia e del livello di sensibilità.
  - `write` – Scrittura di un nuovo documento, specificando nome file, contenuto testuale, tipo e sensibilità.
  - `delete` – Eliminazione di un documento esistente, indicandone l'ID e le relative caratteristiche.
- **Simulazioni di attacco (test IDS/IPS)**:
  - **Simulazione DoS autenticato**: invio ripetuto di richieste HTTP POST per simulare un attacco di tipo denial-of-service verso il PEP.
  - **Port scanning**: scansione delle porte comprese tra la 3070 e la 3120 verso l'IP del PEP tramite `nmap`.
  - **Payload malevolo (NOOP MIPS)**: invio di un payload UDP contenente una sled NOOP in formato MIPS SGI per attivare le regole Snort.

Il menu interattivo appare come segue:

Menu Operazioni

```
=== Menu Operazioni ===
1) Login
2) Logout
3) Read (lettura)
4) Write (scrittura)
5) Delete (cancellazione)
6) Simula DoS
7) Port scanning
8) Payload NOOP Snort
q) Esci
========================
```

5.3 Output e verifica dei risultati

I risultati delle operazioni eseguite sono stampati a schermo in formato **JSON**, e formattati con `jq` se disponibile nel sistema. Questo consente una più agevole interpretazione delle risposte del PEP, comprese eventuali negazioni o autorizzazioni da parte del PDP.

- In caso di operazione autorizzata, la risposta includerà lo status `"access": "granted"` e i dettagli del documento o dell'azione eseguita.
- In caso di diniego, verrà mostrato `"access": "denied"` con la motivazione.
- Per le simulazioni di attacco, il corretto funzionamento del sistema si verifica analizzando i log di `Snort` e `Splunk`, oltre al possibile blocco dinamico dell'IP da parte del router.

5.4 Esempi di comando interni allo script

Login utente via PEP

```
curl -X POST http://<PEP_IP>:3100/login \
     -H "Content-Type: application/json" \
     -c cookies.txt \
     -d '{"username": "alice", "password": "pwd123"}'
```

Lettura documento autenticata

```
curl -X POST http://<PEP_IP>:3100/request \
     -H "Content-Type: application/json" \
     -b cookies.txt \
     -d '{"operation": "read", "document_type": "Dati Personali",
          "doc_id": 1, "sensibilita": "sensibile"}'
```

6 Conclusioni e sviluppi futuri

Il progetto presentato ([6]) ha dimostrato l'efficacia e la flessibilità del paradigma *Zero Trust* applicato a un'infrastruttura containerizzata. Attraverso l'impiego di container Docker, è stato simulato un ambiente realistico e controllato, popolato da client con differenti livelli di fiducia (internal, external e wifi), tutti soggetti a un flusso rigoroso di autenticazione e autorizzazione centralizzata.
L'infrastruttura ha integrato componenti fondamentali di sicurezza come un **PEP** (Policy Enforcement Point) dotato di interfaccia `Flask`, un **PDP** (Policy Decision Point) responsabile delle decisioni di accesso, un motore di logging e monitoraggio basato su **Splunk**, un **IDS** come **Snort** per la rilevazione di minacce, e un **firewall dinamico** gestito tramite `iptables`. Grazie a questa architettura modulare, è stato possibile simulare e testare scenari di attacco e di normale fruizione, verificando la corretta applicazione delle policy di sicurezza, la segmentazione del traffico, e l'efficacia delle contromisure attivate.

Contributi principali del progetto:

- Progettazione e realizzazione di un'architettura Zero Trust interamente containerizzata e replicabile;
- Implementazione di meccanismi di controllo accessi basati su attributi (ABAC), reputazione delle reti e livelli di fiducia;
- Automazione della risposta a eventi di sicurezza attraverso l'integrazione tra Splunk, Snort e moduli di enforcement dinamico;
- Sviluppo di un sistema di testing interattivo per validare scenari reali di accesso e attacco, con misurazione dell'efficacia delle policy.

Sviluppi futuri

A partire dai risultati ottenuti, sono emerse numerose opportunità di miglioramento e approfondimento, che delineano i possibili sviluppi futuri:

- **Integrazione di sistemi di autenticazione avanzati**: introduzione di meccanismi di identità federata o basati su certificati digitali (es. OAuth2, OpenID Connect, SAML), per rafforzare la componente di autenticazione.
- **Analisi intelligente dei log con ML/AI**: adozione di modelli di *machine learning* per l'analisi dei log generati da Snort e PEP, al fine di rilevare pattern anomali o attacchi zero-day in maniera proattiva.
- **Estensione a reti distribuite multi-host**: distribuzione dei container su più nodi fisici, per testare l'architettura in uno scenario di rete geograficamente dislocata, con orchestratori come `Kubernetes`.
- **Ottimizzazione delle prestazioni**: valutazione quantitativa della latenza introdotta dal ciclo decisionale PEP–PDP e dell'impatto delle policy sulle prestazioni complessive (throughput, tempi di risposta, carico di rete).
- **Integrazione con sistemi SIEM e orchestratori SOAR**: per automatizzare in modo ancora più esteso il rilevamento, la correlazione e la risposta agli eventi di sicurezza.

Conclusione

In un contesto in cui le minacce informatiche sono sempre più dinamiche, pervasive e sofisticate, l'approccio Zero Trust si dimostra una strategia robusta ed efficace per costruire architetture resilienti. Il progetto ha evidenziato come una segmentazione fine del traffico, un controllo rigoroso basato su attributi contestuali e un monitoraggio continuo possano costituire un fondamento solido per la sicurezza in ambienti moderni, containerizzati e distribuiti.
La strada tracciata da questo lavoro non rappresenta un punto d'arrivo, bensì un punto di partenza per esplorare soluzioni sempre più intelligenti, adattive e integrate, in linea con i principi di una sicurezza informatica responsabile.

Riferimenti bibliografici

[1] *Docker Documentation*. Accesso: 29 giugno 2025. Docker Inc. URL: `https://docs.docker.com/`.

[2] *Snort Documentation*. Accesso: 29 giugno 2025. Cisco Systems. URL: `https://www.snort.org/documents`.

[3] *Splunk Documentation*. Accesso: 29 giugno 2025. Splunk Inc. URL: `https://docs.splunk.com/Documentation/Splunk/latest`.

[4] *Squid Proxy Server Documentation*. Accesso: 29 giugno 2025. Squid-cache.org. URL: `http://www.squid-cache.org/Doc/`.

[5] Netfilter Team. *iptables Tutorial and Howto*. Accesso: 29 giugno 2025. URL: `https://netfilter.org/documentation/index.html`.

[6] Micol Zazzarini et al. *Repository GitHub del progetto*. 2025. URL: `https://github.com/LauraFe01/AdvCybersecProj` (visitato il giorno 29/06/2025).