

Elaborato Network Security



Intrusion Detection e Analisi del Traffico di Rete tramite l'IDS Suricata

Danilo Romano (matr. M63001542)

Valerio Maietta (matr. M63001407)

A.A. 2022-23

Prof. Simon Pietro Romano

Sommario

Elaborato Network Security.....	1
Sommario.....	2
Introduzione.....	3
Cenni Teorici	4
Cos'è un IDS?	4
Classificazione degli IDS	6
Architettura del Sistema di Rilevamento.....	9
Composizione del Sistema	9
Client e l'esecuzione degli scripts	13
Script Legittimo (test_legitimate.py)	13
Script Malevolo (test_malicious.py)	15
L'IDS Suricata e la sua configurazione	21
Configurazione di Suricata nel nostro Elaborato	22
Configurazione delle regole in Suricata	25
Generazione dei Log	29
Eve.json	29
Fast.log.....	30
Suricata.log	31
Analisi dei Log	31
Filebeat	32
Integrazione con Elasticsearch e Kibana	34
Elasticsearch: Archiviazione e Indicizzazione dei Log	34
Kibana: Analisi e Visualizzazione dei Dati	34
Implementazione con Docker	36
Workflow dell'Interazione	37
Vantaggi dell'Integrazione	38
Manuale di Installazione.....	38
Esecuzione Pratica dell'elaborato.....	41

Introduzione

La crescente complessità delle infrastrutture di rete e l'aumento esponenziale delle minacce informatiche hanno reso indispensabile l'adozione di strumenti avanzati per la protezione delle reti. Questo progetto si propone di implementare un sistema integrato per il rilevamento delle intrusioni (IDS) utilizzando tecnologie open-source e moderne architetture basate su container.

L'obiettivo principale è quello di dimostrare come Suricata, uno degli IDS più potenti e flessibili, possa essere utilizzato per monitorare e analizzare il traffico di rete in tempo reale. Suricata non agisce in isolamento, ma è supportato da Filebeat per la raccolta e il trasferimento dei log, Elasticsearch per l'indicizzazione e l'analisi dei dati, e Kibana per la visualizzazione interattiva e intuitiva degli eventi di rete. L'orchestrazione di queste componenti viene gestita tramite Docker e Docker-Compose, garantendo una configurazione rapida, isolata e scalabile.

Il sistema implementato consente di:

- Monitorare il traffico di rete e individuare anomalie o attacchi in tempo reale.
- Raccogliere, organizzare e indicizzare i log generati da Suricata per un'analisi approfondita.
- Visualizzare i dati raccolti attraverso dashboard interattive, che facilitano l'interpretazione e il rilevamento delle minacce.

Per dimostrare l'efficacia del sistema, sono stati sviluppati e utilizzati due script Python: uno per simulare traffico legittimo e l'altro per generare attacchi malevoli. Questo approccio pratico consente di testare il sistema in scenari reali e di verificarne la precisione nel rilevare minacce e nel distinguere tra traffico normale e sospetto.

L'integrazione di Suricata con la ElasticStack (ElasticSearch e Kibana) rappresenta una soluzione completa per chiunque desideri implementare un sistema di

monitoraggio della rete moderno e scalabile. Quanto introdotto verrà analizzato nel dettaglio nelle pagine successive.

Cenni Teorici

Questo capitolo fornisce una panoramica teorica sui concetti fondamentali legati alla sicurezza informatica, concentrandosi in particolare sul ruolo degli Intrusion Detection System (IDS).

Cos'è un IDS?

Un IDS viene definito nel **RFC 4949** del 2007 come *“un processo o un sottosistema, implementato via software o hardware, che automatizza i compiti di monitoraggio di eventi che capitano in una rete di calcolatori e li analizza al fine di trovare problemi di sicurezza”*, oppure, *“un sistema di allarme di sicurezza per determinare ingressi non autorizzati”*.

Un **Intrusion Detection System (IDS)** si basa su un'architettura logica composta da tre componenti principali: i **sensori**, gli **analizzatori** e l'**interfaccia utente**. Questi elementi lavorano insieme per raccogliere dati, analizzarli e presentare all'utente informazioni utili per rispondere ad eventuali intrusioni. Vediamoli singolarmente:

Sensori

I sensori rappresentano la prima linea di azione di un IDS, responsabili della raccolta dei dati grezzi provenienti da diverse fonti. Questi dati costituiscono la base per il rilevamento delle intrusioni. I sensori possono monitorare una varietà di sorgenti dati, tra cui:

- **Pacchetti di rete:** Analisi dei dati in transito attraverso la rete, per identificare traffico sospetto.

- **File di log:** Controllo delle registrazioni dei sistemi per rilevare attività anomale o non autorizzate.
- **System call traces:** Monitoraggio delle chiamate al sistema operativo effettuate dalle applicazioni, utile per individuare comportamenti non conformi.

In un'implementazione tipica, un IDS può disporre di più sensori posizionati strategicamente per coprire tutte le aree critiche di un sistema o di una rete.

Analizzatori

Gli analizzatori prendono in ingresso i dati raccolti dai sensori e li elaborano per determinare se si è verificata un'intrusione. Questa elaborazione si basa su tecniche come il rilevamento basato su firme o su anomalie, descritte in precedenza.

Nel caso in cui venga identificata un'attività sospetta, l'analizzatore produce un output che contiene:

- **Un report dettagliato:** Include le prove raccolte che indicano la natura dell'intrusione.
- **Eventuali azioni intraprese:** In sistemi configurati come IPS, possono essere registrate azioni preventive, come il blocco di un IP.

In molte implementazioni moderne, i sensori e gli analizzatori sono integrati in un unico componente per migliorare le prestazioni e ridurre la complessità.

Interfaccia Utente

L'interfaccia utente, o componente manager, rappresenta il punto di contatto tra l'operatore e il sistema IDS. Attraverso questa interfaccia, l'operatore può:

- **Visualizzare gli avvisi** generati dagli analizzatori, con dettagli sull'intrusione rilevata.

- **Ricevere notifiche** in tempo reale per rispondere tempestivamente agli incidenti di sicurezza.
- **Gestire la configurazione** del sistema, come la definizione delle regole di rilevamento, l'aggiunta di nuove firme o l'impostazione di soglie per la rilevazione delle anomalie.

Un'interfaccia ben progettata è essenziale per consentire agli operatori di agire rapidamente e comprendere facilmente lo stato della rete o del sistema monitorato.

Proprio per questo, nel nostro elaborato è stato utilizzato **Kibana** come interfaccia utente. Kibana consente una visualizzazione efficace dei dati generati da **Suricata** e **Filebeat**, permettendo di visualizzare in modo intuitivo gli avvisi. Con Kibana ci è stato possibile creare visualizzazioni e dashboard personalizzate, filtrare i dati in modo dinamico e accedere rapidamente alle informazioni necessarie per rispondere alle minacce in modo tempestivo.

Classificazione degli IDS

La classificazione viene fatta in base alla tipologia di analyzer o in base ai sensori. Per quanto riguarda la **classificazione in base ai sensori**, possiamo dividere gli IDS in tre famiglie:

- **Network-based IDS (NIDS):**

Questi sistemi operano intercettando il traffico che attraversa la rete. Analizzano i pacchetti in tempo reale e identificano schemi che corrispondono a minacce conosciute, come attacchi DDoS, malware o tentativi di accesso non autorizzati.

- **Host-based IDS (HIDS):**

I sistemi HIDS, invece, si concentrano sulle attività interne di un singolo

dispositivo. Monitorano file di log, modifiche ai file di sistema e altre attività locali per individuare comportamenti sospetti o non autorizzati.

- **IDS distribuiti o ibridi:**

è un'unione degli approcci precedenti, dunque, combina informazioni locali dell'host e sulla rete al fine di rilevare attività sospette.

Possiamo inoltre effettuare un'ulteriore **classificazione basata stavolta sulla tipologia di analizzatori**; in questo caso avremo IDS di tipo:

- **Rule Based**

Gli IDS rule-based (o signature-based IDS) utilizzano un insieme di regole, o firme, predefinite per rilevare attività sospette. Queste regole sono configurate per identificare comportamenti specifici e noti, come determinati tipi di attacchi informatici (ad esempio, tentativi di exploit, malware o scansioni di porte). Il principio su cui si basa questo tipo di IDS è il confronto del traffico in ingresso o delle attività del sistema con una libreria di "firme" di attacchi precedenti.

Quando un pacchetto o un'azione coincide con una firma nota, il sistema attiva un avviso. Le firme possono includere stringhe di codice, sequenze di pacchetti o comportamenti che corrispondono a vulnerabilità precedentemente documentate.

Se un attaccante sta cercando di sfruttare una vulnerabilità di tipo SQL Injection, un IDS rule-based, come Suricata, può rilevare il tentativo confrontando i pacchetti in ingresso con una firma che descrive i caratteristici pattern di un attacco SQL Injection (ad esempio, parole chiave come OR 1=1 o DROP TABLE).

➤ **Vantaggi**

- Alta precisione nella rilevazione di attacchi noti.
- Minimo numero di falsi positivi, poiché il sistema cerca corrispondenze specifiche con modelli noti.

➤ **Svantaggi**

- Non è in grado di rilevare attacchi nuovi o sconosciuti, poiché si basa solo su firme predefinite.
- Minore flessibilità nel rilevamento di minacce mai viste prima.

▪ **Anomaly-based**

Gli IDS anomaly-based si basano sull'idea di monitorare il traffico di rete o le attività del sistema confrontandole con un "comportamento normale" predefinito. In questo approccio, l'IDS costruisce un modello di comportamento legittimo e segnala ogni deviazione significativa da tale modello come una potenziale intrusione. Questo tipo di IDS è più flessibile rispetto ai rule-based, poiché non necessita di una libreria di attacchi noti, ma si concentra su deviazioni rispetto al normale funzionamento del sistema.

Ad esempio, un attacco DDoS (Denial of Service) potrebbe non avere una firma predefinita, ma un IDS basato su anomalie potrebbe rilevare un piccolo anomalo nel volume del traffico, che è significativamente maggiore rispetto al normale utilizzo della rete. In questo caso, il sistema segnala l'anomalia come un potenziale attacco.

➤ **Vantaggi**

- È in grado di rilevare attacchi nuovi o sconosciuti, poiché non si basa su firme predefinite.

- Maggiore flessibilità nel rilevamento di minacce mai viste prima.

➤ **Svantaggi**

- Maggiore probabilità di falsi positivi, poiché qualsiasi deviazione dal comportamento "normale" viene segnalata come sospetta, anche se si tratta di attività legittime.
- Richiede una fase di addestramento accurata per definire il comportamento "normale"

Architettura del Sistema di Rilevamento

L'architettura del sistema di rilevamento delle intrusioni è progettata per offrire una soluzione robusta e scalabile per monitorare la rete e i sistemi al fine di individuare e rispondere a potenziali minacce. Il sistema si avvale di diverse tecnologie, tra cui **Suricata** come IDS di rete, **Filebeat** per il monitoraggio dei log, e **Kibana** come interfaccia per la visualizzazione e l'analisi dei dati raccolti. Il sistema è completamente containerizzato utilizzando **Docker**, garantendo così una gestione semplificata, scalabilità e isolamento delle componenti. Di seguito viene descritta in dettaglio l'architettura del sistema

Composizione del Sistema

L'architettura del sistema è composta da una serie di componenti modulari che lavorano insieme per raccogliere, analizzare e visualizzare i dati di rete e i log di sistema. I principali componenti del sistema sono i seguenti:

- **Suricata (IDS di rete)**

Suricata rappresenta il cuore del sistema di rilevamento delle intrusioni.

Come **Intrusion Detection System (IDS)**, Suricata monitora il traffico di rete in tempo reale, esaminando i pacchetti per individuare attività sospette o

dannose. Suricata è configurato per generare file di log alla rilevazione di comportamenti malevoli. In particolare, troviamo:

1. *fast.log*: Log in formato semplice e leggibile degli alert
2. *eve.json*: Log dettagliati in formato JSON di tutti gli eventi
3. *suricata.log*: Log di sistema di Suricata stesso (errori, avvii, etc.)

- **Filebeat (Raccolta e invio dei log)**

Filebeat è impiegato per raccogliere i log generati da Suricata e per inviarli ad **Elasticsearch**. Questo strumento è essenziale per tracciare gli eventi di sicurezza, visualizzare gli avvisi e analizzare gli incidenti rilevati. È possibile decidere da quale file di log leggere i dati da inoltrare attraverso il campo *"filebeat.inputs"* presente nel file di configurazione *filebeat.yml*.

- **Elasticsearch (Indice e ricerca dei dati)**

Elasticsearch funge da motore di ricerca e archivio per i dati raccolti da Filebeat. I log e gli avvisi generati da Suricata e Filebeat vengono indicizzati e archiviati in Elasticsearch, il quale permette di eseguire ricerche veloci e analisi sui dati. Elasticsearch supporta anche la scalabilità del sistema, consentendo di gestire grandi volumi di dati in tempo reale.

- **Kibana (Visualizzazione e analisi dei dati)**

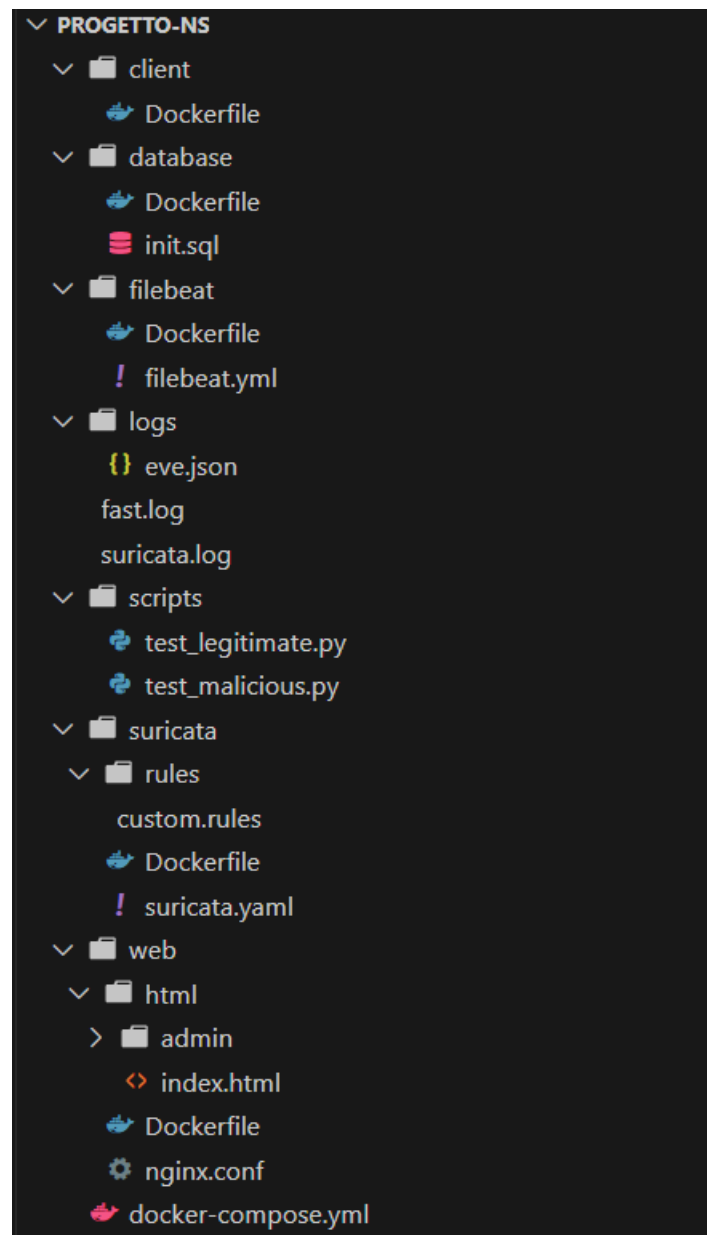
Kibana è la piattaforma di visualizzazione dei dati, utilizzata per creare dashboard interattive e report. Kibana ci consente di visualizzare gli avvisi generati da Suricata attraverso l'ausilio di diverse tipologie di visualizzazioni. L'interfaccia utente di Kibana è fondamentale per la gestione dei dati raccolti, l'identificazione delle minacce e la risposta agli incidenti.

- **Docker (Containerizzazione)**

L'intero sistema è **containerizzato** utilizzando **Docker**, che garantisce un ambiente isolato e facilmente replicabile per ciascun componente. Ogni

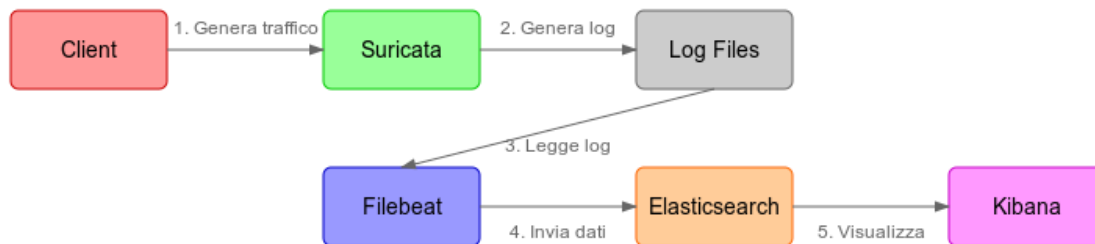
servizio (Suricata, Filebeat, Elasticsearch, Kibana) viene eseguito all'interno di container separati, che possono essere gestiti, aggiornati e scalati indipendentemente l'uno dall'altro. La containerizzazione tramite Docker permette una gestione più efficiente del sistema e facilita l'automazione del deployment.

Riassumendo, questa è la panoramica generale del nostro progetto:



1. Panoramica Generale Progetto

Mentre, l'immagine che segue ha l'obiettivo di presentare il processo di funzionamento del nostro progetto:



2. Processo di Funzionamento

Nella immagine soprastante si può notare il processo generale del nostro elaborato; Il client genera traffico (malevolo o non) tramite i due script python, suricata monitora il traffico e se matcha una delle regole, genera dei log, che vengono letti da Filebeat che, a sua volta, li invia a Elasticsearch e infine Kibana li visualizza.

Client e l'esecuzione degli scripts

Il client nel progetto ha il compito di eseguire i due script Python presenti nella cartella "scripts": uno per simulare attività legittime e uno per simulare attacchi malevoli.

Script Legittimo (test_legitimate.py)

```
1 import requests
2 import time
3 import psycopg2
4
5 def test_web_server():
6     """Effettua richieste HTTP legittime al web server"""
7     print("Eseguo richieste legittime al web server...")
8     for _ in range(5):
9         try:
10             response = requests.get('http://web')
11             print(f"Status code: {response.status_code}")
12             time.sleep(1)
13         except Exception as e:
14             print(f"Errore nella richiesta HTTP: {e}")
15
16 def test_database():
17     """Effettua query legittime al database"""
18     print("Eseguo query legittime al database...")
19     try:
20         # psycopg2.connect() crea una connessione al database PostgreSQL
21         conn = psycopg2.connect(
22             dbname="testdb",
23             user="testuser",
24             password="testpass",
25             host="db"
26         )
27         # Crea un cursore per eseguire query
28         cur = conn.cursor()
29
30         cur.execute("SELECT * FROM users;")
31
32         # Recupera tutti i risultati
33         results = cur.fetchall()
34         print(f"Risultati query: {results}")
35         cur.close()
36         conn.close()
37     except Exception as e:
38         print(f"Errore nella connessione al database: {e}")
39
40 if __name__ == "__main__":
41     test_web_server()
42     test_database()
```

Questo script è progettato per simulare attività di rete legittime, sia verso il server web che verso il database PostgreSQL. Ecco il dettaglio delle sue funzionalità:

1. Funzione test_web_server():

- La funzione invia cinque richieste HTTP al server web (`http://web`), utilizzando il modulo `requests`.
- Ogni richiesta è una chiamata GET e il codice di stato della risposta (`response.status_code`) viene stampato per confermare che il server stia rispondendo correttamente. C'è un ritardo di 1 secondo (`time.sleep(1)`) tra ciascuna richiesta per simulare un comportamento umano naturale.

2. Funzione test_database():

- La funzione tenta di connettersi a un database PostgreSQL tramite il modulo `psycopg2` utilizzando le credenziali di connessione (`dbname`, `user`, `password`, `host`).
- Una volta stabilita la connessione, esegue una query SQL per recuperare tutti i dati dalla tabella `users`.
- I risultati della query vengono stampati per verificare che la connessione e la query siano eseguite correttamente.
- Al termine, la connessione e il cursore vengono chiusi per evitare perdite di risorse.

Script Malevolo (test_malicious.py)

```
1 import requests
2 import time
3 import subprocess # Per eseguire comandi shell (nmap)
4 import threading # Per gestire thread paralleli (DDoS)
5 import urllib.parse # Per codificare caratteri speciali nelle URL
6
7 def simulate_unauthorized_access():
8     """Simula tentativi di accesso non autorizzato"""
9     print("\nSimulazione accessi non autorizzati...")
10    malicious_urls = [
11        'http://web:80/admin',
12        'http://web:80/login?username=admin&password=1234',
13        'http://web:80/config'
14    ]
15
16    # Intestazione HTTP personalizzata
17    headers = {
18        'User-Agent': 'MaliciousScanner/1.0'
19    }
20
21    # Tenta ogni URL tre volte
22    for url in malicious_urls:
23        for _ in range(3):
24            try:
25                print(f"Tentativo di accesso a {url}")
26                response = requests.get(url, headers=headers)
27                print(f"Status code: {response.status_code}")
28                time.sleep(1)
29            except Exception as e:
30                print(f"Errore: {e}")
31
32 def simulate_port_scan():
33     """Simula una scansione delle porte"""
34     print("\nSimulazione port scanning...")
35     try:
36         for port in [80, 8080, 443, 22]:
37             print(f"Scanning porta {port} su 172.18.0.2")
38             subprocess.run(["nmap", "-sS", "-p", str(port), "web"],
39                             check=True, capture_output=True)
40             # check=True solleva un'eccezione se il comando fallisce
41             time.sleep(1)
42     except Exception as e:
43         print(f"Errore durante il port scanning: {e}")
```

4. Script Malevolo pt.1

```

1  def simulate_sql_injection():
2      """Simula tentativi di SQL injection"""
3      print("\nSimulazione SQL injection...")
4
5      base_url = 'http://web:80/login'
6      payloads = [
7          "username=' OR '1'='1' --&password=any",
8          "username=admin' UNION SELECT * FROM users--&password=any",
9          "username=' OR 1=1 #&password=any",
10         "username=admin' OR 1=1;--&password=any"
11     ]
12     for payload in payloads:
13         try:
14             url = f'{base_url}?{payload}'
15             print(f"Tentativo SQL injection: {url}")
16             response = requests.get(url)
17             print(f"Status code: {response.status_code}")
18             time.sleep(1)
19         except Exception as e:
20             print(f"Errore durante SQL injection: {e}")
21
22 def single_ddos_request(url):
23     """Singola richiesta per il DDoS"""
24     try:
25         requests.get(url)
26     except:
27         pass
28
29 def simulate_ddos():
30     """Simula un attacco DDoS"""
31     print("\nSimulazione attacco DDoS...")
32     url = 'http://web:80/'
33
34     # Crea 100 thread per fare richieste simultanee
35     threads = []
36     for _ in range(100):
37         # Ogni thread esegue la funzione single_ddos_request
38         thread = threading.Thread(target=single_ddos_request, args=(url,))
39         threads.append(thread) # Salva il thread nella lista
40         thread.start() # Avvia il thread
41
42     # Ripete l'attacco 3 volte
43     for _ in range(3):
44         for thread in threads:
45             thread.join() # Aspetta che il thread finisca
46             time.sleep(1)
47         print(f"Invia 100 richieste simultanee a {url}")
48
49 if __name__ == "__main__":
50     print("Inizio dei test di sicurezza...")
51
52     # Esegui ogni tipo di attacco separatamente
53     simulate_port_scan()
54     time.sleep(2)
55     print("\n" + "="*50 + "\n")
56
57     simulate_unauthorized_access()
58     time.sleep(2)
59     print("\n" + "="*50 + "\n")
60
61     simulate_sql_injection()
62     time.sleep(2)
63     print("\n" + "="*50 + "\n")
64
65     simulate_ddos()
66
67     print("\nTest completati.")

```

5. Script Malevolo pt.2

Questo script simula una serie di attacchi malevoli, sfruttando richieste HTTP, scansioni di rete e tentativi di SQL Injection per testare la capacità del sistema di rilevare minacce. Analizziamo ciascuna funzione nello specifico:

1. Funzione *simulate_unauthorized_access()*:

Questa funzione esegue tentativi di accesso non autorizzato al server web, simulando comportamenti malevoli come scansioni e exploit di URL riservati.

Nello specifico:

- **Lista degli URL che vogliamo tentare di accedere:**

La lista `malicious_urls` contiene endpoint sensibili comunemente bersagliati dagli attaccanti:

- `/admin`: tipico endpoint per accedere all'area di amministrazione.
- `/login?username=admin&password=1234`: simula un tentativo di login con credenziali deboli.
- `/config`: tentativo di accedere ai file di configurazione.

- **Intestazione HTTP personalizzata:**

Viene specificato un User-Agent personalizzato (*MaliciousScanner/1.0*) per simulare un tool malevolo che esegue scansioni o attacchi.

- **Ripetizione dei tentativi:**

Ogni URL viene richiesto tre volte per insistere sugli endpoint potenzialmente non protetti.

- **Esecuzione:**

- Il modulo "requests" invia una richiesta HTTP GET per ciascun URL.
- Il codice di stato della risposta viene stampato, fornendo un'indicazione sul successo o sul blocco dell'accesso.

L'obiettivo dell'attacco è quello di scoprire endpoint non protetti o debolmente configurati e verificare se il sistema rileva questi tentativi come sospetti.

2. Funzione *simulate_port_scan()*

Simula una scansione delle porte (port scanning), che è spesso la fase iniziale di un attacco per identificare servizi in ascolto su un host. Nello specifico:

- **Lista delle porte:**

Viene eseguita una scansione di porte comuni:

- 80: HTTP.
- 8080: HTTP alternativo, spesso usato per server di test.
- 443: HTTPS.
- 22: SSH.

- **Utilizzo di Nmap:**

- Usa il comando nmap con l'opzione -sS, che esegue una scansione SYN stealth:
 - Invio di un pacchetto SYN a ciascuna porta.
 - Se il server risponde con un SYN-ACK, la porta è aperta.
 - Se risponde con un RST, la porta è chiusa.
- L'opzione -p specifica la porta da scansionare, e "web" è il target della scansione.

- **Output e gestione errori:**

- Il comando viene eseguito tramite *subprocess.run*, catturando l'output e gestendo eventuali errori.
- L'intervallo di un secondo tra le scansioni evita di saturare immediatamente il sistema.

L'obiettivo dell'attacco è quello di identificare porte aperte e servizi esposti sul server web. Suricata rileva questa attività come un comportamento malevolo grazie alla apposita regola nel file "*custom.rules*".

3. Funzione *simulate_sql_injection()*:

Simula tentativi di SQL Injection inviando payload malevoli a un endpoint di login. Nello specifico:

Base URL:

`http://web:80/login` è l'endpoint che riceve i payload.

- **Payload malevoli:**

Una serie di stringhe SQL malevole viene inviata come query string. Esempi:

- `"username=' OR '1'='1' --&password=any"`: bypassa l'autenticazione con una condizione sempre vera.
- `"username=admin' UNION SELECT * FROM users--&password=any"`: tenta di estrarre dati unendo la tabella users ai risultati della query.
- `"username=' OR 1=1 #&password=any"`: un'altra variante di condizione sempre vera.

- **Meccanismo di invio:**

- Ogni payload viene inviato tre volte per aumentare la probabilità di successo.
- Il payload viene anche codificato in URL (`urllib.parse.quote`) per verificare se il sistema filtra correttamente i caratteri speciali.

- **Esecuzione:**

- Per ogni payload, vengono inviate due richieste: una versione normale e una codificata.
- I codici di stato delle risposte sono registrati per analizzare il comportamento del server.

L'obiettivo dell'attacco è quello di forzare il server a eseguire query SQL arbitrarie e verificare se il sistema di rilevamento riconosce i pattern di SQL Injection.

4. Funzione *simulate_ddos()*:

Simula un attacco DDoS (Distributed Denial of Service) per sovraccaricare il server. Nello specifico:

- **URL target:**

http://web:80/ è l'endpoint bersaglio delle richieste.

- **Creazione dei thread:**

- Vengono creati 100 thread, ciascuno incaricato di inviare una singola richiesta HTTP GET.
- Ogni thread esegue la funzione `single_ddos_request`.

- **Esecuzione simultanea:**

- Tutti i thread vengono avviati simultaneamente per generare un alto volume di traffico in breve tempo.
- Il processo viene ripetuto tre volte con un intervallo di 1 secondo.

L'obiettivo dell'attacco è quello di saturare le risorse del server e interrompere la sua operatività.

L'IDS Suricata e la sua configurazione

Suricata è un potente sistema di rilevamento delle intrusioni (IDS), progettato per monitorare il traffico di rete e identificare potenziali minacce alla sicurezza. Si tratta di un **IDS di tipo rule-based** e **NIDS (Network Intrusion Detection System)**, il che significa che si basa principalmente su un set di regole predefinite per rilevare attività dannose o sospette all'interno del traffico di rete. Suricata è in grado di monitorare in tempo reale il flusso di pacchetti, analizzando vari protocolli e contenuti per identificare attacchi conosciuti, vulnerabilità, e comportamenti anomali.

Essendo un sistema **rule-based**, Suricata utilizza un vasto database di regole per rilevare gli attacchi. Queste regole sono scritte in un formato che definisce specifici pattern nel traffico di rete che, quando riscontrati, vengono segnalati come potenziali minacce.

Configurazione di Suricata nel nostro Elaborato

Il file *suricata.yml* contiene le impostazioni principali per la configurazione dell'IDS. Di seguito c'è il nostro **suricata.yml**:

```
1  %YAML 1.1
2  ---
3  # Definizione delle reti
4  vars:
5    address-groups:
6      HOME_NET: '[172.18.0.0/16]' # La nostra rete Docker che vogliamo monitorare
7      EXTERNAL_NET: 'any' # Qualsiasi rete esterna
8
9  # Percorso delle regole e file delle regole
10 default-rule-path: /etc/suricata/rules
11 rule-files:
12   - custom.rules
13
14 # Configurazione degli output
15 outputs:
16   - fast:
17       enabled: yes
18       filename: /var/log/suricata/fast.log
19       append: yes
20   - eve-log:
21       enabled: yes
22       filetype: regular
23       filename: /var/log/suricata/eve.json
24       types:
25         - alert
26         - http
27
28 app-layer:
29   protocols:
30     http:
31       enabled: yes
32       libhttp:
33         default-config:
34           personality: IDS
35
36 #Per gli eventi relativi a suricata (file suricata.log)
37 logging:
38   default-log-level: debug
39   outputs:
40     - file:
41         enabled: yes
42         level: debug
43         filename: /var/log/suricata/suricata.log
44
```

6. *suricata.yml*

❖ Variabili di rete (vars)

Il file di configurazione inizia con la definizione delle **variabili di rete**. Le variabili `HOME_NET` e `EXTERNAL_NET` sono configurate in modo da riflettere l'architettura di rete specifica.

- **HOME_NET** definisce la rete "interna" da proteggere ed è configurato come l'intervallo di indirizzi IP interni. Usiamo `172.18.0.0/16` perché è la rete Docker che abbiamo configurato nel `docker-compose.yml`.
- **EXTERNAL_NET** definisce le reti considerate "esterne" ed è configurato come *any*, il che significa che tutto il traffico proveniente da indirizzi esterni (non appartenenti a `HOME_NET`) verrà considerato come potenzialmente sospetto

❖ Percorsi delle regole (default-rule-path e rule-files)

Un'altra parte importante della configurazione riguarda le regole di rilevamento, che sono definite nel parametro **rule-files**. Queste regole vengono utilizzate per identificare tentativi di intrusione o attività dannose nel traffico di rete:

- **default-rule-path** indica la directory in cui Suricata cerca le regole, in questo caso `/etc/suricata/rules`.
- **rule-files** include il file **custom.rules**, che contiene regole personalizzate per il rilevamento di attacchi specifici, aggiungendo un ulteriore livello di personalizzazione al sistema di rilevamento delle intrusioni.

La configurazione delle custom rules verrà spiegata dettagliatamente in seguito.

❖ Output dei log (outputs)

La configurazione di **Suricata** prevede anche diverse opzioni per la gestione degli output, ossia come i risultati dell'analisi del traffico vengono registrati e archiviati. Nel file di configurazione, si trovano le seguenti sezioni relative agli output:

- **fast**: Attiva il log in formato fast, che contiene informazioni sintetiche sugli eventi rilevati da Suricata, come gli allarmi generati. Questo file di log, **fast.log**, è scritto nella directory `/var/log/suricata/` e consente di avere una visione rapida degli eventi, utile per un monitoraggio tempestivo.
- **eve-log**: Attiva il log in formato **EVE JSON**, che fornisce informazioni più dettagliate sugli eventi. In questo caso, vengono registrati eventi di tipo **alert** e **http**, ed il file viene salvato come **eve.json**. Questo formato è facilmente integrabile con altre piattaforme di analisi, come **Elasticsearch**, per una visualizzazione e analisi avanzata tramite strumenti come **Kibana**.

❖ Livello applicativo (app-layer)

Suricata offre anche la possibilità di analizzare il traffico a livello applicativo per identificare attacchi più specifici, come quelli diretti alle applicazioni web. Questa sezione consente a Suricata di analizzare il traffico HTTP, abilitando l'analisi dei protocolli applicativi. Il parametro **personality** è impostato su **IDS**, il che significa che Suricata funzionerà in modalità di rilevamento delle intrusioni (non di prevenzione), monitorando il traffico HTTP per eventuali attacchi come **SQL injection**, o altre vulnerabilità.

❖ Livello di log e destinazione (logging)

Il file di configurazione include anche una sezione dedicata al logging interno di Suricata stesso (utile per il debugging), che permette di definire il livello di dettaglio dei log generati dal sistema.

Nel nostro caso, il **livello di log** è impostato su **debug**, il che significa che verranno registrati messaggi molto dettagliati per consentire una diagnosi approfondita di eventuali problemi. I log saranno scritti nel file **suricata.log**.

Configurazione delle regole in Suricata

Le regole di **Suricata** sono un elemento cruciale per il rilevamento degli attacchi, poiché determinano quali eventi e pattern devono essere monitorati nel traffico di rete. Le regole definiscono i segnali di allarme che il sistema deve generare quando identifica attività sospette, come tentativi di intrusione, scansioni di porte o attacchi a livello applicativo. In questo capitolo, si esplorerà la configurazione delle regole personalizzate per il progetto, contenute nel file **custom.rules**, e come queste regole siano utilizzate da Suricata per rilevare i vari tipi di attacchi. Il file **custom.rules** utilizzato nel progetto include diverse regole per il rilevamento di tentativi di accesso non autorizzato, vulnerabilità a livello di applicazione e attacchi a rete e sistema. Ogni regola è progettata per identificare un tipo specifico di comportamento sospetto nel traffico di rete, generando un avviso (alert) quando viene rilevata un'anomalia.

Questo è il nostro custom.rules:

```
1 # Regole per rilevare tentativi di accesso all'area admin
2 alert http any any -> $HOME_NET any (msg:"Admin Access Attempt Detected"; flow:established,to_server; http.uri; content:"/admin"; sid:1000001; rev:1;)
3
4 # Regole per rilevare tentativi di login sospetti
5 alert http any any -> $HOME_NET any (msg:"Suspicious Login Attempt Detected"; flow:established,to_server; http.uri; content:"/login"; content:"username=admin"; within:50; sid:1000002; rev:1;)
6
7 # Regole per rilevare accessi a configurazioni
8 alert http any any -> $HOME_NET any (msg:"Configuration Access Attempt Detected"; flow:established,to_server; http.uri; content:"/config"; sid:1000003; rev:1;)
9
10 # Regola per rilevare port scanning
11 alert tcp any any -> $HOME_NET any (msg:"Potential Port Scan Detected"; flags:S; threshold: type threshold, track by_src, count 5, seconds 60; sid:1000004; rev:1;)
12
13 # Regole multiple per SQL Injection
14 alert http any any -> $HOME_NET any (msg:"SQL Injection Attempt - UNION Detected"; flow:established,to_server; http.uri; content:"UNION"; nocase; sid:1000005; rev:1;)
15 alert http any any -> $HOME_NET any (msg:"SQL Injection Attempt - OR Condition"; flow:established,to_server; http.uri; content:"OR"; content:"-"; distance:0; within:2; nocase; sid:1000006; rev:1;)
16 alert http any any -> $HOME_NET any (msg:"SQL Injection Attempt - Comment"; flow:established,to_server; http.uri; content:"-"; sid:1000007; rev:1;)
17 alert http any any -> $HOME_NET any (msg:"SQL Injection Attempt - Single Quote"; flow:established,to_server; http.uri; content:"'"; sid:1000008; rev:1;)
18
19 # Regola per rilevare attacchi DDoS
20 alert tcp any any -> $HOME_NET any (msg:"Potential DDoS Attack Detected"; flags:S; threshold: type both, track by_src, count 100, seconds 5; sid:1000009; rev:1;)
21
```

7. custom.rules

✓ Regole per il rilevamento di accessi non autorizzati

Una delle regole più comuni riguarda il rilevamento di tentativi di accesso a sezioni sensibili di un'applicazione web, come l'area di amministrazione. Un esempio di regola per il rilevamento di tentativi di accesso all'area **admin** è quella descritta nel nostro file `custom.rules`.

In questa regola:

- **alert:** indica che viene generato un avviso quando la regola è soddisfatta. Nella regola, *any any* indica che il traffico può provenire da **qualsiasi indirizzo IP e porta sorgente**. Per quanto riguarda la destinazione, *\$HOME_NET any* significa che il traffico è diretto verso **qualsiasi indirizzo IP** nella rete *\$HOME_NET* e **qualsiasi porta di destinazione**, senza restrizioni.
- **flow:established,to_server:** in Suricata specifica che la regola deve analizzare solo i flussi TCP già **stabiliti** (cioè, dopo il completamento del three-way handshake) e diretti **verso il server**. Questo restringe l'analisi alle connessioni valide e alle richieste provenienti dai client, riducendo falsi positivi e concentrandosi sul traffico rilevante.
- **http.uri; content:"/admin":** questa parte cerca nel traffico HTTP l'URI che contiene `"/admin"`, indicando un tentativo di accesso all'area di amministrazione.
- **sid:1000001:** è un identificatore univoco per la regola.

Questa regola viene attivata ogni volta che un tentativo di accesso alla directory `/admin` viene effettuato nel traffico verso una risorsa della rete definita come *HOME_NET* (variabile che abbiamo definito in `suricata.yaml` come `172.18.0.0/16`).

✓ Rilevamento di login sospetti

Un'altra regola riguarda il rilevamento di tentativi di login sospetti, in particolare quando viene tentato l'accesso con il nome utente `"admin"`, che è un obiettivo comune per gli attaccanti. La regola descritta nel nostro file rileva quando nel

traffico HTTP viene trovato un tentativo di login con il nome utente "admin" nella URI /login. Il parametro **within:50** assicura che il contenuto "username=admin" venga trovato entro 50 byte dalla posizione della URI /login, garantendo che il login sospetto venga associato correttamente alla pagina di accesso.

✓ Rilevamento di vulnerabilità a livello di applicazione

Le regole per il rilevamento di vulnerabilità a livello di applicazione, come le **SQL Injection**, sono altrettanto fondamentali. Queste regole cercano segni tipici di attacchi come **UNION SQL Injection** o **Comment SQL Injection** nel traffico HTTP. Le regole per **SQL Injection** cercano vari pattern nel traffico HTTP che indicano tentativi di manipolazione di query SQL. Le parole chiave come **UNION**, **OR**, **--** (commento SQL) e **'** (apostrofo, spesso usato per iniettare codice SQL) sono monitorate per identificare i tentativi di sfruttamento delle vulnerabilità applicative.

✓ Rilevamento di scansioni di porte

Un altro tipo di attacco che può essere rilevato da Suricata è il **port scanning**, una tecnica usata per identificare porte vulnerabili su un sistema di destinazione. Nella regola scritta nel nostro file custom.rules, il traffico TCP con il flag **SYN** (flag S, indicante una connessione iniziale) viene monitorato. Il campo threshold in Suricata serve per controllare quante volte una regola deve attivarsi prima di generare un alert. In questo caso:

- **track by_src**: Specifica che il monitoraggio delle occorrenze deve avvenire separatamente per **ogni indirizzo IP sorgente**. Ogni IP sorgente che supera la soglia di connessioni a porte diverse verrà rilevato.
- **count 5**: Indica che la regola si attiva se ci sono **5 tentativi di connessione** da un singolo indirizzo IP sorgente.

- **seconds 60**: Specifica che la regola considera solo i tentativi che si verificano nell'arco di **60 secondi**.

Se un singolo indirizzo sorgente tenta di stabilire più di 5 connessioni in un intervallo di 60 secondi, la regola segnala una possibile scansione di porte.

✓ **Rilevamento di attacchi DDoS**

Infine, una delle regole più importanti riguarda il rilevamento degli attacchi **Distributed Denial of Service (DDoS)**, che mirano a sopraffare un sistema con un volume elevato di traffico.

Questa regola monitora il traffico TCP con flag SYN e segnala un possibile attacco DDoS se 100 pacchetti vengono ricevuti dallo stesso indirizzo sorgente entro 5 secondi. Questo tipo di attività è tipico di un attacco DDoS, in cui un grande numero di richieste è inviato simultaneamente al server per sovraccaricarlo.

Generazione dei Log

La generazione dei log è un aspetto cruciale in un sistema di **Intrusion Detection System (IDS)** come Suricata, in quanto consente di registrare e monitorare gli eventi rilevati durante il processo di analisi del traffico di rete. I log generati forniscono informazioni dettagliate sugli incidenti di sicurezza, inclusi i tentativi di intrusione, gli attacchi rilevati e altre anomalie, che possono essere successivamente utilizzati per analisi forensi, audit e risposta agli incidenti. Nel caso specifico del sistema configurato, Suricata genera tre file di log principali: **eve.json**, **fast.log**, e **suricata.log**. Ognuno di questi file ha uno scopo specifico e fornisce dettagli differenti sugli eventi monitorati.

Eve.json

Il file **eve.json** è uno dei principali file di log utilizzati da Suricata per registrare informazioni dettagliate sugli eventi di intrusione rilevati. Il formato JSON consente una facile elaborazione e analisi dei dati, poiché è un formato strutturato e leggibile sia per l'uomo che per le macchine. Il file **eve.json** può contenere eventi di diverso tipo, tra cui allarmi (alerts), traffico HTTP, DNS, flussi di rete e altre informazioni utili per l'analisi dei dati.

Nel nostro progetto, il file **eve.json** è configurato per generare avvisi (alerts) specifici sugli attacchi rilevati, come quelli relativi a tentativi di accesso non autorizzato, vulnerabilità applicative e scansioni di porte. Alcuni esempi di informazioni che si possono trovare nel file **eve.json** includono:

- Dettagli sull'attacco (ad esempio, tipo di attacco, indirizzo sorgente, indirizzo di destinazione, e il contenuto del pacchetto)
- Identificatori univoci per ogni allarme, come il **sid** (Suricata ID) e la **rev** (revisione, versione della regola)

- Dettagli del flusso di rete (ad esempio, informazioni sulla connessione HTTP)
- Timestamps che indicano quando l'evento è stato rilevato.

Il file **eve.json** è utile per l'integrazione con strumenti di visualizzazione e monitoraggio, come **Kibana**, che possono analizzare e visualizzare i dati in tempo reale.

Fast.log

Il file **fast.log** è un log di tipo "**fast output**", che fornisce un registro conciso e ad alta velocità degli eventi rilevati. È progettato per produrre rapidamente informazioni sugli allarmi, con un formato di output semplice che consente di monitorare velocemente i principali eventi di sicurezza senza doversi immergere nei dettagli più complessi. Mentre il file **eve.json** offre una panoramica dettagliata degli attacchi, il file **fast.log** è più orientato alla registrazione rapida degli eventi, ad esempio gli attacchi rilevati o le anomalie nel traffico di rete.

In un ambiente di produzione, **fast.log** è utile per operazioni quotidiane di monitoraggio e per l'integrazione con sistemi di allerta in tempo reale, dove la rapidità di risposta è fondamentale. Alcuni esempi di voci che si trovano nel **fast.log** includono:

- Timestamp dell'evento.
- Tipo di traffico (es. HTTP, TCP, DNS).
- Messaggio dell'allarme, che descrive l'evento o il tentativo di intrusione.
- Indirizzi IP di origine e destinazione coinvolti nell'attacco.

Questo file di log è essenziale per rispondere rapidamente agli eventi in tempo reale, soprattutto quando si cerca di intervenire tempestivamente in caso di attacchi in corso.

Suricata.log

Il file **suricata.log** è il file di log principale utilizzato da Suricata per registrare informazioni generali sullo stato del sistema IDS. A differenza di **eve.json** e **fast.log**, che si concentrano sulla registrazione degli eventi di intrusione, il file **suricata.log** registra i dettagli relativi al funzionamento e alla configurazione di Suricata stesso. Questo include informazioni di debug, errori e messaggi di stato che sono utili per gli amministratori di sistema e gli analisti della sicurezza.

Alcuni esempi di voci che si possono trovare nel **suricata.log** includono:

- Dettagli di avvio e shutdown di Suricata.
- Messaggi relativi alla configurazione del sistema.
- Eventuali errori nel processo di analisi o problemi tecnici con i sensori e gli analizzatori.
- Messaggi di stato relativi al flusso di traffico o alla gestione delle regole.

Il file **suricata.log** è fondamentale per il monitoraggio del funzionamento interno di Suricata, la diagnostica e la risoluzione dei problemi.

Analisi dei Log

Una volta che i log sono generati e registrati, è possibile eseguire diverse operazioni di analisi. Strumenti come **Kibana**, che si integrano perfettamente con i file di log in formato JSON, permettono di visualizzare e filtrare gli eventi in modo semplice ed efficace. Ad esempio, gli amministratori possono cercare rapidamente per tipo di attacco, indirizzo IP di origine o timestamp, per identificare incidenti di sicurezza.

Inoltre, la gestione centralizzata dei log e l'analisi automatizzata degli stessi possono contribuire a ridurre il tempo di risposta agli incidenti, migliorando la capacità di rilevamento e la protezione complessiva del sistema.

Filebeat

Filebeat è uno strumento progettato per raccogliere e inoltrare i log generati da diverse fonti, tra cui i log di **Suricata**, verso sistemi centralizzati di analisi come **Elasticsearch**. La sua funzionalità principale consiste nel monitorare i file di log in tempo reale, raccogliere gli eventi generati e trasmetterli in modo sicuro e scalabile a destinazioni come Elasticsearch, dove i dati possono essere visualizzati tramite **Kibana**.

Nel contesto del sistema descritto, **Filebeat** è configurato per raccogliere i log generati da **Suricata**, come il file **eve.json**, che contiene informazioni dettagliate sugli eventi di intrusione e attacco rilevati. Filebeat monitora costantemente questo file di log, catturando ogni nuovo evento e inoltrandolo a una destinazione di elaborazione centrale, in questo caso **Elasticsearch**, per una gestione e un'analisi più efficienti.

```
1 filebeat.inputs:
2   - type: log
3     enabled: true
4     paths:
5       - /var/log/suricata/eve.json # il file che filebeat deve monitorare
6     json.keys_under_root: true # per avere i campi json come campi di primo livello
7     json.add_error_key: true # per avere info su eventuali errori
8     fields:
9       type: suricata
10    fields_under_root: true # come "json.keys_under_root" ma per i campi "fields" (custom)
11
12 output.elasticsearch:
13   hosts: ['elasticsearch:9200'] # dove inviare i log raccolti
14   index: 'suricata-%{+yyyy.MM.dd}' # indice in elasticsearch
15
16 setup.kibana:
17   host: 'kibana:5601'
18
19 # Configurazione del template Elasticsearch
20 setup.template.name: 'suricata'
21 setup.template.pattern: 'suricata-*'
22 setup.template.enabled: true
23 setup.template.overwrite: true # permette di sovrascrivere template esistenti in caso avessero stesso nome
24
25 setup.ilm.enabled: false # disabilita Index Lifecycle Management
26 logging.level: info
27
28 # Aggiunge metadati ai documenti
29 processors:
30   - add_host_metadata: ~ # info sull'host
31   - add_cloud_metadata: ~ # info sul cloud provider (se presente)
32   - add_docker_metadata: ~ #info sui container docker
33
```


Come possiamo vedere, la configurazione di **Filebeat** include vari aspetti chiave:

- **Input dei Log:** Il file di configurazione specifica i percorsi di file da monitorare, in questo caso **eve.json** di Suricata, che viene trattato come un log JSON. Questo consente di gestire i dati in modo strutturato, aggiungendo metadati rilevanti (come il tipo di log, indicato come suricata) e gestendo eventuali errori nel processo di parsing.
- **Output su Elasticsearch:** Filebeat è configurato per inviare i log raccolti a un'istanza di **Elasticsearch**, all'indirizzo `elasticsearch:9200`. I dati vengono inviati sotto forma di indici giornalieri, con il formato `suricata-%{+yyyy.MM.dd}`. Questo approccio consente di organizzare i dati in indici separati per ogni giorno, facilitando la gestione e l'accesso alle informazioni storiche.
- **Integrazione con Kibana:** La configurazione include anche la connessione a **Kibana** (`kibana:5601`), un'interfaccia grafica che permette di visualizzare i dati raccolti da Filebeat. Kibana facilita l'analisi dei log attraverso dashboard personalizzabili, consentendo agli operatori di monitorare in tempo reale gli eventi di sicurezza.
- **Template di Elasticsearch:** È stato configurato un template specifico per gli indici di Suricata in Elasticsearch. Il template definisce le impostazioni necessarie per gestire i dati provenienti da Suricata, come il formato degli indici e altre configurazioni avanzate di indicizzazione. L'uso di template permette una gestione più coerente e ottimizzata dei dati nel sistema Elasticsearch.

Integrazione con Elasticsearch e Kibana

L'integrazione di **Suricata** con **Elasticsearch** e **Kibana** rappresenta una delle componenti principali del sistema di rilevamento delle intrusioni descritto. Questa configurazione permette di centralizzare i log generati da Suricata, analizzarli in modo efficiente e visualizzarli attraverso interfacce intuitive.

Nell'elaborato, il deployment di Elasticsearch e Kibana è stato semplificato utilizzando le loro immagini Docker ufficiali, configurate attraverso il file *docker-compose.yml*. Questo approccio non solo velocizza l'implementazione ma garantisce anche un ambiente flessibile e facilmente replicabile.

Elasticsearch: Archiviazione e Indicizzazione dei Log

Elasticsearch è un motore di ricerca e analisi distribuito che consente di archiviare e interrogare i dati in modo scalabile. Nel contesto di Suricata, Elasticsearch riceve i log inviati da **Filebeat**, li indicizza e li rende disponibili per ricerche complesse e analisi avanzate.

Ogni log generato da Suricata viene trasformato in un documento JSON, che viene archiviato in indici all'interno di Elasticsearch. La configurazione prevede l'uso di indici giornalieri, ad esempio *suricata-2025.01.05*, che consentono una gestione più efficiente e una segmentazione temporale dei dati.

Kibana: Analisi e Visualizzazione dei Dati

Kibana è l'interfaccia utente di ElasticStack che consente di esplorare i dati archiviati in Elasticsearch attraverso dashboard, grafici e strumenti di analisi interattivi.

Con Kibana, è possibile:

- **Monitorare gli avvisi generati da Suricata:** Dashboard personalizzate mostrano il numero di intrusioni rilevate, i tipi di attacco e la loro origine.
- **Analizzare eventi specifici:** Grazie alla ricerca avanzata, è possibile filtrare i log per SID, tipo di attacco, indirizzi IP coinvolti o periodi temporali.
- **Identificare tendenze:** Grafici e metriche temporali aiutano a comprendere l'evoluzione degli eventi di sicurezza nella rete monitorata.
- **Creare notifiche e avvisi personalizzati:** Kibana permette di impostare regole per notificare gli operatori quando si verificano determinati tipi di attacco.

Esempio di Utilizzo

Una dashboard in Kibana può mostrare in tempo reale:

- Gli attacchi rilevati suddivisi per tipo (es. SQL injection, port scanning, DDoS).
- La mappa geografica degli indirizzi IP di origine degli attacchi.
- I SID delle regole che hanno generato più avvisi.

Grazie a questa integrazione, il sistema di rilevamento degli attacchi non solo identifica le minacce, ma le rende facilmente comprensibili e gestibili attraverso strumenti avanzati di analisi e visualizzazione.

Implementazione con Docker

Sia Elasticsearch che Kibana sono stati distribuiti tramite container Docker, utilizzando immagini preconfigurate. Questa scelta offre numerosi vantaggi, come la portabilità e la semplicità di configurazione.

```
1  elasticsearch:
2    image: docker.elastic.co/elasticsearch/elasticsearch:7.9.3
3    container_name: elasticsearch
4    environment:
5      - discovery.type=single-node # singolo nodo perché è un ambiente di test
6      - 'ES_JAVA_OPTS=-Xms512m -Xmx512m' # impostiamo la memoria minima e massima a 512MB
7      - xpack.security.enabled=false # disabilitiamo la sicurezza per semplificare la configurazione ed evitare problemi di autenticazione con Filebeat e Kibana
8    ports:
9      - '9200:9200'
10   networks:
11     security_network:
12       ipv4_address: 172.18.0.5
13   healthcheck:
14     test: ['CMD', 'curl', '-f', 'http://localhost:9200']
15     interval: 10s
16     timeout: 10s
17     retries: 5
18
19   kibana:
20     image: docker.elastic.co/kibana/kibana:7.9.3
21     container_name: kibana
22     ports:
23       - '5601:5601'
24     environment:
25       - ELASTICSEARCH_HOSTS=http://elasticsearch:9200
26     depends_on:
27       elasticsearch:
28         condition: service_healthy
29     networks:
30       security_network:
31         ipv4_address: 172.18.0.6
```

9. parte del docker-compose di elasticsearch e kibana

Di seguito è riportata una sintesi delle configurazioni utilizzate nel file docker-compose.yml:

- **Elasticsearch:**
 - **Immagine:** docker.elastic.co/elasticsearch/elasticsearch:7.9.3
 - **Configurazione:**
 - Modalità nodo singolo (discovery.type=single-node) per un setup semplificato.
 - Heap Java limitato a 512 MB per ottimizzare l'utilizzo delle risorse.
 - X-Pack Security disattivato (xpack.security.enabled=false) per facilitare l'integrazione con Kibana.

- **Collegamento alla rete:** È stato utilizzato un indirizzo IP statico nella rete Docker personalizzata, garantendo una comunicazione stabile con Kibana e Filebeat.
- **Healthcheck:** Il sistema verifica l'operatività del container tramite richieste periodiche HTTP (curl).
- **Kibana:**
 - **Immagine:** docker.elastic.co/kibana/kibana:7.9.3
 - **Configurazione:**
 - Collegamento diretto a Elasticsearch tramite la variabile di ambiente ELASTICSEARCH_HOSTS=http://elasticsearch:9200.
 - Interfaccia web accessibile sulla porta 5601.
 - Avvio subordinato alla disponibilità di Elasticsearch, gestito tramite la direttiva depends_on con una condizione di healthcheck.
 - **Rete e indirizzo IP statico:** Kibana è stato configurato per utilizzare un IP fisso nella rete Docker, semplificando il routing.

Workflow dell'Interazione

1. **Generazione dei log:** Suricata produce log dettagliati, come **eve.json**, che contengono informazioni sugli eventi rilevati (ad esempio, tentativi di accesso non autorizzato o scansioni delle porte).
2. **Raccolta tramite Filebeat:** I log vengono raccolti da Filebeat, che li pre-elabora e li invia a Elasticsearch.
3. **Archiviazione in Elasticsearch:** I log vengono indicizzati e resi disponibili per analisi e ricerche.
4. **Visualizzazione in Kibana:** Gli operatori accedono a Kibana per monitorare, analizzare e gestire gli eventi di sicurezza.

Vantaggi dell'Integrazione

- **Centralizzazione dei log:** Tutti gli eventi di sicurezza sono archiviati in un unico sistema, semplificando la gestione.
- **Analisi in tempo reale:** L'architettura permette di analizzare gli eventi quasi istantaneamente dopo la loro rilevazione.
- **Flessibilità e personalizzazione:** Gli strumenti offrono una grande varietà di opzioni per personalizzare dashboard, report e notifiche.
- **Automazione e proattività:** L'integrazione consente di automatizzare il monitoraggio degli eventi, permettendo di rispondere più rapidamente alle minacce.

Manuale di Installazione

Questo capitolo fornisce le istruzioni dettagliate per l'installazione e l'avvio del sistema di monitoraggio della sicurezza di rete basato su Docker. L'intero progetto è contenuto in un repository Git, e utilizza **Docker Compose** per orchestrare i diversi container necessari.

Struttura del Sistema

Il sistema è composto dai seguenti componenti:

- **Server Web (Nginx):** Fornisce un ambiente web simulato per il test.
- **Database (PostgreSQL):** Gestisce i dati generati durante i test.
- **IDS di Rete (Suricata):** Rileva intrusioni e attacchi nel traffico di rete.
- **Stack ELK (Elasticsearch e Kibana):** Archivia e visualizza i log di sicurezza.
- **Client di Test:** Simula attività legittime e maligne per valutare il sistema.

Prerequisiti

Prima di procedere con l'installazione, assicurarsi di soddisfare i seguenti requisiti:

1. **Software richiesto:**

- **Docker:** Assicurarsi che Docker sia installato e configurato correttamente.
- **Docker Compose:** Verificare che Docker Compose sia installato.

2. Hardware:

- Almeno **4 GB di RAM** disponibili.

3. Porte disponibili:

- **8080** per il server web.
- **5601** per Kibana.
- **9200** per Elasticsearch.

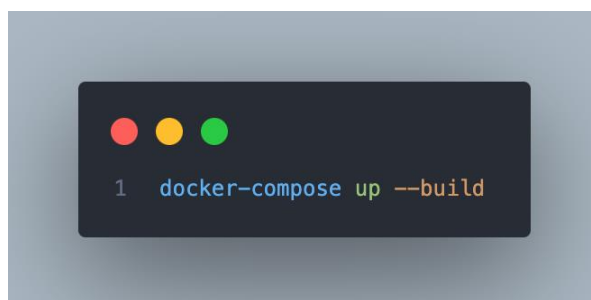
Installazione

1. Clonare il repository Git: Scaricare il progetto utilizzando Git:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. It displays two lines of code: 1 git clone https://github.com/ValerioMaietta/Progetto-NS.git and 2 cd Progetto-NS.

```
1 git clone https://github.com/ValerioMaietta/Progetto-NS.git
2 cd Progetto-NS
```

2. Avviare il sistema Utilizzare Docker Compose per avviare i container:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. It displays one line of code: 1 docker-compose up --build.

```
1 docker-compose up --build
```

- Il comando esegue il build di tutte le immagini necessarie e avvia i container definiti nel file docker-compose.yml.

- Durante il primo avvio, il processo potrebbe richiedere alcuni minuti per scaricare le immagini e configurare i container.
3. **Verificare lo stato:** Al termine dell'avvio, verificare che tutti i container siano in esecuzione con il comando "docker ps":

Configurazione di Kibana

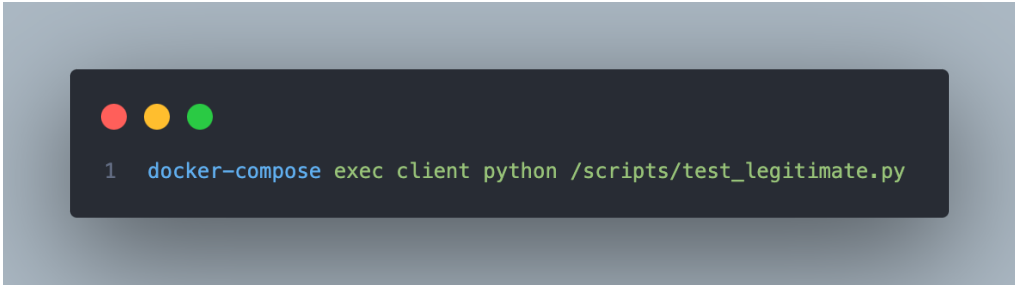
Per visualizzare i dati generati da Suricata, è necessario configurare Kibana:

- Accedere a Kibana aprendo un browser e collegandosi a **<http://localhost:5601>**.
- Creare un pattern di indice:
 - Navigare su **Stack Management > Index Patterns**.
 - Creare un pattern utilizzando suricata-* come nome dell'indice.
- Configurare le visualizzazioni:
 - Cliccare su **Visualize**.
 - Creare nuove visualizzazioni per monitorare gli alert e i log generati dal sistema.
- Salvare le visualizzazioni come parte di una dashboard per una consultazione rapida.

Esecuzione dei Test di Sicurezza

Per verificare il corretto funzionamento del sistema, eseguire test simulati di attività legittime e maligne utilizzando gli script forniti.

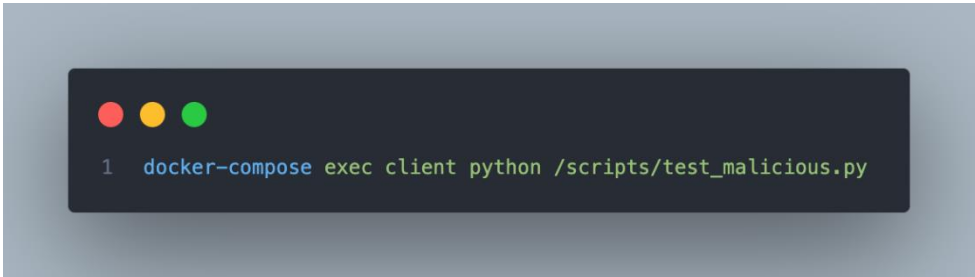
1. **Test delle attività legittime:** Dal container client, eseguire il seguente comando:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The command `1 docker-compose exec client python /scripts/test_legitimate.py` is entered in a light green monospace font.

```
1 docker-compose exec client python /scripts/test_legitimate.py
```

Questo script simula traffico legittimo, verificando che il sistema non generi falsi positivi.

2. **Test delle attività maligne: Simulare** attacchi eseguendo:

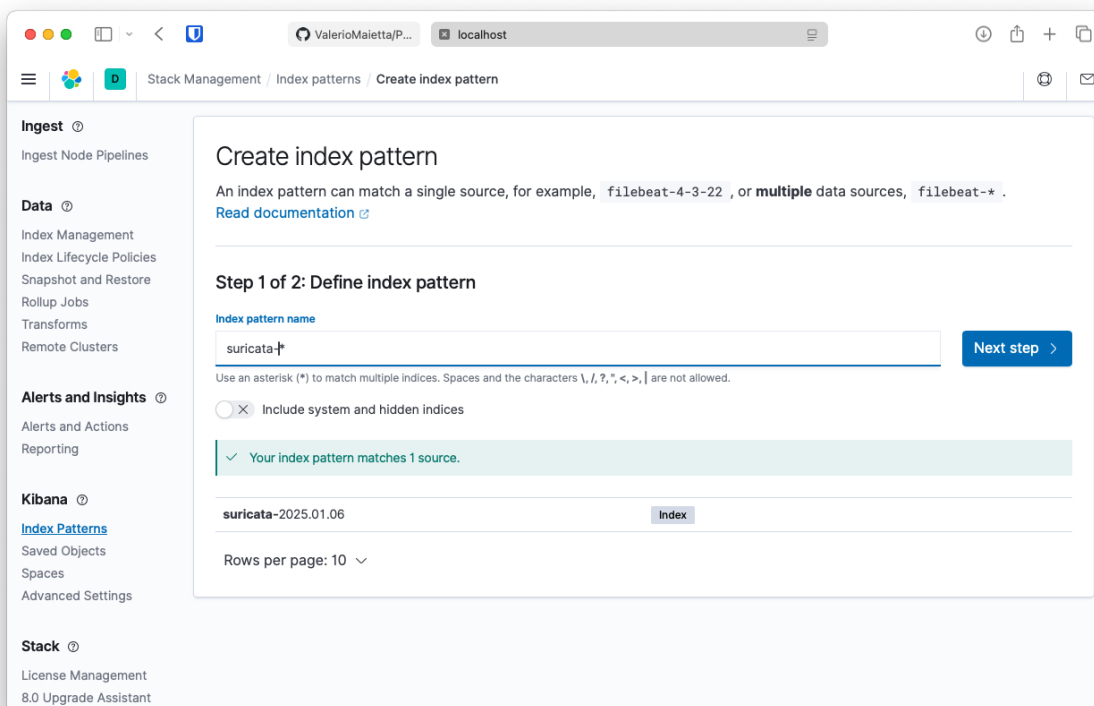
A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The command `1 docker-compose exec client python /scripts/test_malicious.py` is entered in a light green monospace font.

```
1 docker-compose exec client python /scripts/test_malicious.py
```

Questo script genera traffico malevolo, come tentativi di SQL Injection o attacchi di port scanning. Gli alert corrispondenti saranno visibili in Kibana.

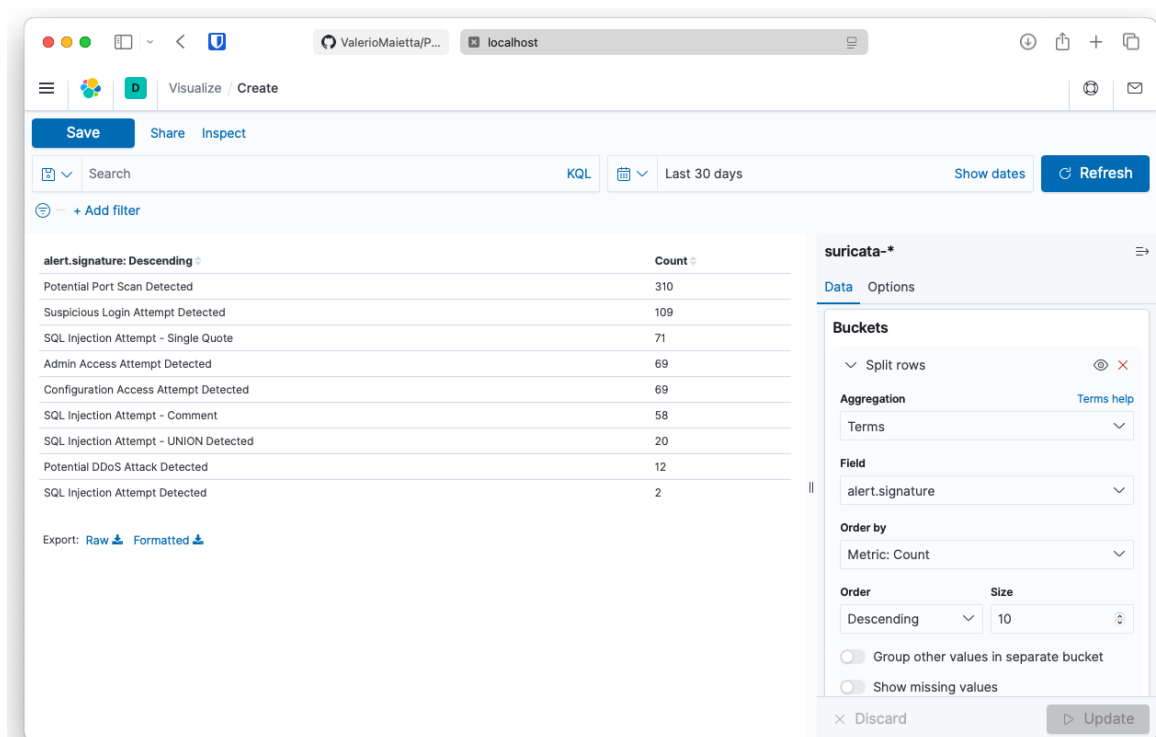
Esecuzione Pratica dell'elaborato

Una volta generati sufficienti alert, è possibile visualizzarli sull'interfaccia grafica messa a disposizione da Kibana. Per fare ciò, dobbiamo andare su localhost:5601/. Dopodiché dobbiamo creare un index pattern. Per farlo dobbiamo andare nel menù laterale e cliccare in basso su “Stack Management”, dopodiché cliccare su “Index Patterns” ed infine su “Create index pattern”.



Nel nostro caso sarà “*suricata-**” in modo tale da fare il match con qualsiasi alert di qualsiasi giorno (dato che gli alert vengono prodotti come “*suricata- + {data odierna}*”). Successivamente selezioniamo “@timestamp” come “Time field” e l’indice verrà correttamente creato.

Ora è il momento di creare le visualizzazioni con le quali andremo a formare la Dashboard di Kibana. Per fare ciò andiamo sul menù laterale e clicchiamo su “Visualize”, poi su “Create new visualization” e successivamente cliccare sul tipo di visualizzazione. A scopo di esempio useremo la visualizzazione “Data Table”. Nel nostro esempio abbiamo inserito il campo “alert.signature”, ovvero il nome dell’alert generato e il campo “Count”, che equivale al numero di alert generati per quel tipo di alert. A questo punto ci basterà solo salvare la visualizzazione.



Analogamente potremmo andare a creare più tipi di visualizzazioni che verranno usate infine per creare la Dashboard. Per fare ciò, clicchiamo su “Dashboard” dal menu laterale, poi su “Create new dashboard” ed infine selezioniamo le visualizzazioni che dovranno comporre la dashboard. Di seguito un esempio:

