



UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

Network Security

Documentazione Progetto Network Security

Anno Accademico 2024/2025

Professore: **Prof. Simon Pietro Romano**
Luigi Guerrera M63001386

Contents

1	Introduzione	1
1.1	Intrusion Detection Systems (IDS)	1
1.2	Suricata	2
1.2.1	File di configurazione Suricata.yaml	3
1.2.2	Suricata rules	5
1.2.3	Utilizzo di Suricata a linea di comando	6
2	Docker Lab	7
2.1	Architettura della rete	7
2.1.1	Router	8
2.1.2	Webserver	9
2.1.3	Attacker	10
2.2	Docker Compose	11
3	Svolgimento laboratorio (Soluzione)	13
3.1	Attacchi	13
3.1.1	Scan	13
3.1.2	XSS	16
3.1.3	DoS	18
3.1.4	SQL Injection	20

Chapter 1

Introduzione

L'obiettivo del presente elaborato consiste nella descrizione del processo di progettazione e configurazione di un Laboratorio in ambiente Docker che mira ad emulare una configurazione di rete semplificata che presenta un nodo (sul quale è in esecuzione un **IDS basato su Suricata**) per il monitoraggio del traffico da e verso il webserver per identificare attacchi comuni in modo da poter reagire con appropriate tecniche di Remediation. Scopo del laboratorio è quello di acquisire esperienza pratica nella configurazione di un Intrusion Detection System per il monitoring della sicurezza di rete.

1.1 Intrusion Detection Systems (IDS)

Un Sistema di Rilevamento delle Intrusioni (**Intrusion Detection System ,IDS**) è un sistema che cerca di individuare le intrusioni, progettato per monitorare il traffico di rete o l'attività del sistema in cerca di comportamenti sospetti o dannosi.

Nell'**RFC 4949** si definisce IDS come "Un servizio di sicurezza che monitora e analizza gli eventi della rete o del sistema al fine di trovare e fornire degli avvisi (warnings), in tempo reale o quasi, relativi a tentativi di accesso a risorse del sistema in maniere non autorizzate."

Le intrusioni di sicurezza sono definite dal RFC come: "Un evento di sicurezza, o una combinazione di multipli eventi di sicurezza, che costituiscono un incidente di sicurezza nel quale un intruder ottiene, o cerca di ottenere, accesso al sistema o alle risorse di sistema senza avere l'autorizzazione".

Si possono classificare gli analyzer in base all'approccio con cui opera, in particolare si distinguono gli IDS tra :

- **Anomaly based**, che individua comportamenti che si discostano da un comportamento normale, quindi è una strategia che può sollevare molto spesso dei falsi positivi.

- **Signature based**, che individua solo quei comportamenti riconosciuti come malevoli sulla base di signature. Non sono, quindi, in grado di riconoscere attacchi di cui non si conosca già la signature.

In pratica gli IDS rilevano potenziali attacchi informatici, violazioni della sicurezza o altre attività anomale e avvisano gli amministratori in tempo reale.

Gli IDS sono importanti per i costi di gestione delle intrusioni, e quindi effettuare azioni tempestive, le quali talvolta permettono di salvare delle aziende, e anche perché quando i sistemi di prevenzione delle intrusioni (IPS) falliscono, allora la seconda linea di difesa sono i sistemi di rilevamento delle intrusioni.

Esistono due principali tipi di IDS:

- quelli che analizzano il traffico di rete (**NIDS**), e
- quelli che monitorano specifici dispositivi o server (**HIDS**).

Possiamo inoltre riferirci ad una qualunque combinazione dei due come ad un IDS distribuito (**DIDS**).

1.2 Suricata

Suricata è Suricata è un IDS open-source network-based e rule-based, un sistema di rilevamento e prevenzione delle intrusioni (IDS/IPS) open-source, particolarmente avanzato e versatile in quanto trova applicazione quando non è possibile installare un agent su un dispositivo, come ad esempio succede nel caso mobile.

Oltre a monitorare il traffico di rete per individuare minacce e attacchi, Suricata offre funzionalità avanzate come l'ispezione del protocollo, il rilevamento delle anomalie e la possibilità di decodificare il traffico in tempo reale. Una delle sue caratteristiche principali è **l'utilizzo di regole personalizzabili che permettono di specificare quali tipi di traffico o comportamenti monitorare**, oltre a supportare l'integrazione con strumenti come **Emerging Threats** per mantenere aggiornato il database delle minacce conosciute. Suricata è ampiamente diffuso poiché altamente performante e scalabile, utilizzabile in una varietà di contesti che spaziano dalle reti domestiche alle reti aziendali.

Uno dei limiti di Suricata è che se lavoriamo con protocolli che prevedono cifratura, come ad esempio succede con HTTPS, abbiamo che tutti i pacchetti sono criptati e quindi non vediamo nulla se non l'IP. In questi casi quindi possiamo usare solo una IP black list.

Suricata può funzionare non solo come IDS ma anche come IPS.

Un **IPS (Intrusion Prevention System)** è un sistema di sicurezza progettato per prevenire e bloccare attacchi o intrusioni dannose all'interno di una rete. A differenza di

un IDS, che si limita a rilevare e segnalare eventuali minacce o comportamenti sospetti, un IPS agisce attivamente per fermare gli attacchi in corso.

1.2.1 File di configurazione Suricata.yaml

Il file **suricata.yaml** permette di configurare Suricata per gestire numerosi aspetti dell'IDS/IPS quali il traffico da monitorare e il modo in cui vengono gestiti gli allarmi.

Inoltre permette di configurare l'ottimizzazione delle prestazioni e di gestire l'integrazione con altri strumenti di monitoraggio.

Un'accurata configurazione di questo file consente a Suricata di operare in modo efficiente e di adattarsi alle necessità specifiche di rete e sicurezza di ciascun ambiente.

Il file di configurazione usato dal container router_suricata è mostrato nel paragrafo 2.1.1, dove viene anche descritta la particolare configurazione utilizzata più nel dettaglio.

Nella sezione **af-packet**, è possibile specificare le interfacce di rete che Suricata deve monitorare e i relativi parametri di acquisizione del traffico. In particolare si possono specificare

- **interface**, che specifica quali interfacce di rete utilizzare per catturare il traffico,
- **defrag**, che se abilitato indica a Suricata di occuparsi della ricostruzione dei pacchetti frammentati ai fini dell'analisi.

Ulteriori parametri quali **threads** e **cluster-type** per bilanciare il carico su CPU multiple, **ring-size** per specificare la buffer size per i pacchetti per ciascun thread, altri parametri **use-mmap** e **tpacket-v3** ottimizzano Suricata nella modalità IPS. Insieme a questi per abilitare il funzionamento di Suricata come IPS si settano anche i valori **drop** e **reject** di **block**. Quindi in questo caso (IPS) la configurazione di questa sezione sarebbe:

```
af-packet:
- interface: eth0
  ...
  defrag: yes
  block:
    drop: yes
    reject: yes
```

Nella sezione **rule-files**, è possibile specificare quali file di regole devono essere caricati da Suricata. Serve specificare anche il path dove vanno cercate tali regole all'interno del FS della macchina

```
default-rule-path: /etc/suricata/rules
rule-files:
- suricata.rules
- fileAlternativo.rules
```

Le variabili di rete (**vars**) vengono usate per definire reti interne, esterne o specifiche, consentendo alle regole di essere flessibili. In particolare è pratica comune definire le reti **HOME_NET**, che definisce l'insieme degli indirizzi IP considerati parte della rete locale, ed **EXTERNAL_NET**, che definisce gli indirizzi esterni. Solitamente si imposta il valore di **EXTERNAL_NET** in modo che tutto ciò che non è parte di **HOME_NET** viene considerato **EXTERNAL_NET**, quindi i valori che più comunemente si trovano sono **any** oppure **"!\$HOME_NET"**. Ad esempio possiamo avere

```
vars:
  address-groups:
    HOME_NET: "[172.18.0.0/16, 10.0.0.0/16]"
    EXTERNAL_NET: "!$HOME_NET"
```

Suricata può essere configurato per riconoscere e analizzare specifici protocolli a vari livelli del modello ISO/OSI attraverso la key-word **app-layer**.

Ai fini del logging si configurano i file dove mantenere i log e attributi per la costruzione dei log quali i tipi di log da salvare. A tal scopo si usa la sezione **outputs**, dove vengono quindi specificati i vari file di log con i rispettivi attributi (ad esempio **filetype**, **filename**, **types**).

```
outputs:
  - eve-log:
      enabled: yes
      filetype: regular
      filename: /var/log/suricata/eve.json
      types:
        - alert:
            payload: yes
            payload-printable: yes
            packet: yes
        - dns:
        - http:
        - tls:
```

Ci sono sezioni che possono essere usate attraverso la sezione **flow** per il performance tuning se si conosce l'hardware a disposizione. In oltre con l'opzione **emergency-recovery** si setta la percentuale di flussi eliminata in caso di esaurimento della memoria.

La sezione **flow-timeouts** sezione controlla i timeout per diversi tipi di flusso, come TCP, UDP, e ICMP.

Un limite di Suricata riguarda il monitoring relativo a protocolli che prevedono cifratura. Nel caso di TLS si può configurare Suricata in modo da raccogliere informazioni dai certificati relativi alle comunicazioni oggetto di analisi, in particolare **certs-log** abilita la registrazione dei certificati TLS e **certs-fingerprint** definisce l'algoritmo di hashing da usare per i fingerprint dei certificati. Si può trovare ad esempio:

```
tls:
```

```
enabled: yes
certs-log: yes
certs-fingerprint: sha1
```

1.2.2 Suricata rules

Le regole Suricata permettono, attraverso la loro struttura flessibile e potente, di rilevare e gestire un'ampia varietà di attacchi a diversi livelli dello stack di rete, dal livello rete al livello applicativo.

Struttura delle regole Suricata

Una regola consiste di 3 componenti distinte:

- **(1) Action** che determina l'azione da compiere quando la regola viene attivata;
- **(2) Header**, che definisce il protocollo da ispezionare, gli indirizzi IP e le porte di mittente e destinatario e la direzione del flusso;
- **(3) Rule options** che definiscono dei vincoli più specifici della regola.

Con un esempio possiamo vedere meglio le suddette parti, ad esempio se consideriamo la regola

```
alert tcp any any -> any 80 (msg:"HTTP traffic detected"; sid:1000001;)
```

1) La prima parola della regola definisce l'azione che Suricata intraprenderà quando la regola viene soddisfatta. Le azioni valide sono:

- **alert**: genera un avviso;
- **drop**: scarta il pacchetto (richiede l'uso come IPS);
- **pass**: ignora il pacchetto;
- **reject**: blocca e invia un messaggio di errore RST/ICMP all'origine.

2) Viene quindi definito il protocollo da monitorare. Subito dopo si specificano indirizzo e porta del mittente e del destinatario. Il primo valore è l'IP che può assumere il valore di un IP specifico, una IP mask oppure il valore any che rappresenta qualsiasi indirizzo IP. Il secondo valore è la porta, anche questa può essere indicata con un numero preciso, un range del tipo 80:443 (che indica tutte le porte nel range tra 80 a 443) oppure può assumere il valore any che indica qualunque porta.

3) Infine vengono specificate le opzioni, in particolare le più comunemente usate sono le seguenti:

- **msg**, per fornire un messaggio descrittivo dell'evento, che viene mostrato quando la regola genera un allarme, e **sid**, l'identificativo univoco della regola.

- **content**, utilizzato per specificare i pattern di dati quali stringhe o byte che Suricata deve cercare all'interno del payload del pacchetto.
- **PCRE** che permette di usare espressioni regolari per pattern complessi.
- **flow**, per definire la direzione del flusso della connessione.
- Controllo sui pacchetti in termini dei **flags** del protocollo TCP, della **dsize** ovvero la dimensione del payload e del **ttd** (time to live) dei pacchetti.
- contenuto a livello di protocollo (es.metodo HTTP usato) o l'URL della richiesta.

1.2.3 Utilizzo di Suricata a linea di comando

Per utilizzare Suricata da linea di comando, è necessario avviare l'IDS specificando il file di configurazione e le interfacce di rete da monitorare. Il comando di avvio tipico (nel caso di una interfaccia) è:

```
suricata -c /etc/suricata/suricata.yaml -i eth0 -i eth1
```

Dove -c indica il percorso del file di configurazione ed -i specifica l'interfaccia di rete da monitorare.

Per visualizzare in tempo reale i log generati da Suricata, è possibile utilizzare il comando **tail -f**, che aggiorna costantemente la visualizzazione del file, mostrando in tempo reale gli eventi di sicurezza rilevati da Suricata. Comandi per monitorare il log in tempo reale:

```
tail -f /var/log/suricata/eve.json | jq 'select(.event_type=="alert")'
```

```
tail -f /var/log/suricata/fast.log
```


Chapter 2

Docker Lab

2.1 Architettura della rete

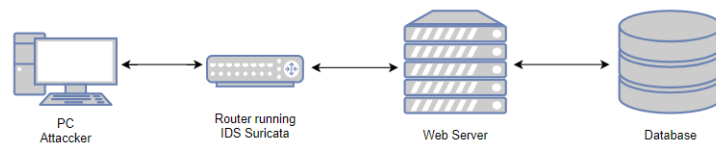


Figure 2.1: Schema della rete

Il laboratorio presenta quattro componenti principali:

- un nodo attaccante (Kali Linux),
- un router con Suricata (Ubuntu),
- un webserver (Ubuntu) e
- un database (MySQL).

```
9098999:~/.ssh$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
7f44938c1907	lab6_webserver	"/start_ws.sh"	About a minute ago	Up About a minute	0.0.0.0:8080->80/tcp, :::8080->80/tcp	webserver
ae84aa226d96	dockersecplayground/kali	"entrypoint.sh shell..."	About a minute ago	Up About a minute		lab6_attacker_1
ed8d92df825d	lab6_suricata	"/start.sh"	About a minute ago	Up About a minute	0.0.0.0:8080->8080/tcp, :::8080->8080/tcp	router_suricata
9c8647999e29	mysql:5.7	"docker-entrypoint.s..."	About a minute ago	Up About a minute	0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp	lab6_db_1

Figure 2.2: Containers

Questi nodi sono collegati attraverso reti virtuali isolate che garantiscono una separazione logica delle comunicazioni. I collegamenti tra i container sono affidati alle seguenti reti:

- Rete `attack_net`: dedicata alla comunicazione tra l'attaccante e il router Suricata, simulando tentativi di attacco che provengono dall'esterno.
- Rete `web_net`: collega il webserver e il router che esegue Suricata. Questa rete simula la comunicazione tra un server che espone servizi web e il router che funge da filtro di sicurezza.
- Rete `internal_net`: Rete privata tra il webserver e il database, isolata dalle altre reti. Il database è accessibile solo dal webserver, impedendo comunicazioni dirette da parte dell'attaccante.

2.1.1 Router

Il container Router si occupa del forwarding dei pacchetti dall'attaccante al web server e viceversa sfruttando le funzionalità di iptables e le due interfacce di rete (**eth0** verso il web server ed **eth1** verso la rete esterna dove risiede l'attaccante). Suricata esegue su questo container in modo da analizzare con facilità tutto il traffico scambiato tra attaccante e web server, dal momento che tutto il traffico passa attraverso questo nodo.

Interfacce:

- `eth0`, con IP 172.18.0.2
- `eth1`, con IP 172.19.0.2

L'implementazione e la configurazione del container sono svolte con i file `LabIDS/suricata/Dockerfile` e `LabIDS/suricata/start.sh`.

IPTables

Le regole di iptables servono per gestire il traffico di rete che passa attraverso un sistema, decidendo quali pacchetti di dati possono essere accettati, rifiutati o modificati. Ogni regola viene applicata in base a tabelle che specificano il comportamento.

Ogni regola di iptables è composta da tabelle. Una tabella rappresenta il contesto in cui la regola viene applicata. Le tabelle più usate sono:

- `filter`, che gestisce l'accettazione o il rifiuto dei pacchetti.
- `nat`, che modifica i pacchetti per l'instradamento o si occupa della traduzione degli indirizzi di rete.

Ogni tabella contiene diverse catene a cui attribuire una regola, condizioni per farla applicare ed azioni da intraprendere.

In particolare l'utilizzo di iptables fatto in questo contesto configura alcune regole di iptables per gestire il traffico di rete attraverso una macchina che funge da router tra due interfacce di rete (`eth0` e `eth1`).

```
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Questa regola appartiene alla tabella nat e alla catena POSTROUTING. La sua funzione è quella di abilitare il NAT (Network Address Translation) per il traffico in uscita dall'interfaccia eth0. L'opzione MASQUERADE nasconde l'IP di origine dei pacchetti provenienti dalla rete interna dietro l'IP dell'interfaccia eth0.

```
iptables -A FORWARD -i eth1 -o eth0 -j ACCEPT
iptables -A FORWARD -i eth0 -o eth1 -j ACCEPT
```

Queste regole appartengono alla catena FORWARD e consentono rispettivamente il traffico che proviene dall'interfaccia eth1 (la rete interna) verso l'esterno attraverso l'interfaccia eth0 (la rete esterna) e viceversa dall'interfaccia eth0 verso l'interfaccia eth1. Quindi consente il routing del traffico dalla rete interna verso l'esterno e dalla rete esterna verso l'interno.

Iptables è uno strumento molto versatile per la gestione delle regole di filtraggio del traffico di rete e, sebbene nel contesto attuale venga usato solo per il routing e il NAT, è comunemente impiegato per costruire **firewall**. Attraverso le sue tabelle e catene, è possibile definire regole che consentano o blocchino il traffico in base a vari criteri come indirizzi IP, porte e protocolli, rendendolo efficace per proteggere sistemi da accessi non autorizzati.

In particolare, in contesti di **remediation**, iptables può essere utilizzato per mitigare attacchi specifici attraverso il blocco di traffico proveniente da indirizzi IP malevoli, chiusura di porte che espongono servizi vulnerabili o limitazione dei tentativi di connessione.

Suricata IDS: configurazione e regole

Il funzionamento di suricata viene configurato attraverso un file di configurazione in yaml convenzionalmente chiamato suricata.yaml e attraverso dei file con estensione .rules specificati nel file di configurazione. Questi insieme specificano il comportamento di Suricata come IDS o IPS.

I file per la configurazione di Suricata e delle sue regole sono contenuti in **/LabIDS/-suricata/suricata.yaml** e **/LabIDS/suricata/suricata.rules**.

Durante lo svolgimento serve aggiungere regole nel file suricata.rules o vanno aggiunti ulteriori file di regole in suricata.yaml affinché si riescano a rilevare i tipi di attacco proposti. Le regole che riescono a rilevare questi tipi di attacchi sono presentati nel capitolo 3, di seguito alla relativa presentazione dell'attacco.

2.1.2 Webserver

Il container Webserver esegue il server apache per poter fornire l'accesso (attraverso l'interfaccia **eth0**) ad un semplice sito web vulnerabile ad alcune tipologie di attacchi. Attraverso una seconda interfaccia di rete **eth1** il nodo può interagire con un nodo dove risiede un database server MySQL.

Interfacce:

- eth0, con IP 172.18.0.3
- eth1, con IP 10.0.0.3

Nella cartella **/LabIDS/webserver/** sono contenuti il Dockerfile file per la generazione del container, lo script di entripoint del container ed il file per la configurazione di apache.

Website

La web application che è stata implementata ad hoc per il laboratorio è molto elementare e presenta poche pagine html e php, in particolare è presente una homepage index.html che permette di raggiungere dal browser alcune pagine di interesse ai fini degli attacchi svolti. I sorgenti di suddette pagine web sono nella cartella **/LabIDS/webserver/web-site/** e sono: **index.html**, **about.html**, **xssVuln.php** e **sqlVuln.php**.

Database MySQL

Il nodo database ospita un database MySQL accessibile attraverso l'interfaccia di rete eth0. L'unico nodo con cui interagisce è il web server.

Interfacce: eth0, con IP 10.0.0.4

```
-- Per fare il login in MySQL
mysql -u user -p

-- Per selezionare il database nel DBServer
USE testdb;

-- Per creare una table nel DB
CREATE TABLE utenti (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL
);

-- Per inserire un valore nella tabella
INSERT INTO utenti (nome) VALUES ('Kvaratskhelia');
```

2.1.3 Attacker

Il nodo attacker, con OS Kali linux in esecuzione, sfruttando l'interfaccia di rete eth0 può connettersi al router al fine di interagire con il web server per inoltrare richieste lecite e anche tentare di implementare attacchi di vario genere.

Interfaccia: eth0, con IP 172.19.0.3

2.2 Docker Compose

```
version: '3.8'
services:
  suricata:
    build: ./suricata
    container_name: router_suricata
    privileged: true
    cap_add:
      - NET_ADMIN # Necessario per gestire le interfacce di rete
    networks:
      web_net:
        ipv4_address: 172.18.0.2
      attack_net:
        ipv4_address: 172.19.0.2
    environment:
      - INTERFACES=eth0 eth1
    ports:
      - "8000:8000" # Se si volesse accedere ai log o statistiche via web

  db:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: root_password
      MYSQL_DATABASE: testdb
      MYSQL_USER: user
      MYSQL_PASSWORD: password
    ports:
      - "3306:3306"
    networks:
      internal_net:
        ipv4_address: 10.0.0.4

  webserver:
    build: ./webserver
    container_name: webserver
    privileged: true
    networks:
      web_net:
        ipv4_address: 172.18.0.3
      internal_net:
        ipv4_address: 10.0.0.3
    depends_on:
```

```
    - suricata
ports:
  - "8080:80"
environment:
  - INTERFACES=eth0 eth1
cap_add:
  - NET_ADMIN

attacker:
  image: dockersecplayground/kali
  stdin_open: true
  networks:
    attack_net:
      ipv4_address: 172.19.0.3
  cap_add:
    - NET_ADMIN
  tty: true # Mantiene il container aperto per interazioni manuali
  depends_on:
    - suricata

networks:
  web_net:
    driver: bridge
    ipam:
      config:
        - subnet: 172.18.0.0/16
  internal_net:
    driver: bridge
    ipam:
      config:
        - subnet: 10.0.0.0/16
  attack_net:
    driver: bridge
    ipam:
      config:
        - subnet: 172.19.0.0/16
```

Chapter 3

Svolgimento laboratorio (Soluzione)

L'avvio e lo svolgimento degli attacchi sono presentati nel file IstruzioniAvvio.

3.1 Attacchi

Il laboratorio consiste nell'eseguire degli attacchi di diverso tipo sul webserver e riuscire a riconfigurare Suricata in modo da identificare e magari reagire ad una serie di tipologie di attacco quali i seguenti:

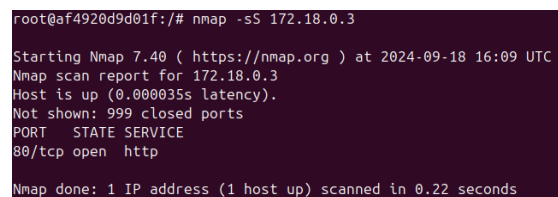
3.1.1 Scan

Gli attacchi Scan consistono nell'analizzare un sistema per identificare servizi attivi, porte aperte o vulnerabilità sfruttabili, spesso usato come fase di preparazione per attacchi più mirati.

Implementazione ed Identificazione dell'attacco

Lo scanning attraverso nmap, con un comando quale

```
nmap -sS 172.18.0.3
```



```
root@af4920d9d01f:/# nmap -sS 172.18.0.3
Starting Nmap 7.40 ( https://nmap.org ) at 2024-09-18 16:09 UTC
Nmap scan report for 172.18.0.3
Host is up (0.000035s latency).
Not shown: 999 closed ports
PORT      STATE SERVICE
80/tcp    open  http
Nmap done: 1 IP address (1 host up) scanned in 0.22 seconds
```

Figure 3.1: AttaccoScan

```
{
  "timestamp": "2024-09-18T16:09:21.189203+0000",
  "flow_id": 531147778073111,
  "in_iface": "eth1",
  "event_type": "alert",
  "src_ip": "172.19.0.3",
  "src_port": 64279,
  "dest_ip": "172.18.0.3",
  "dest_port": 1723,
  "proto": "TCP",
  "pkt_src": "wire/pcap",
  "alert": {
    "action": "allowed",
    "gid": 1,
    "signature_id": 100002,
    "rev": 1,
    "signature": "Nmap TCP Connect scan detected",
    "category": "Attempted Information Leak",
    "severity": 2
  },
  "direction": "to_server",
  "flow": {
    "pkts_toserver": 1,
    "pkts_toclient": 0,
    "bytes_toserver": 58,
    "bytes_toclient": 0,
    "start": "2024-09-18T16:09:21.189203+0000",
    "src_ip": "172.19.0.3",
    "dest_ip": "172.18.0.3",
    "src_port": 64279,
    "dest_port": 1723
  }
}
```

Figure 3.2: LogScan

Alcune possibili regole che si possono usare per rilevare questo tipo di attacco sono le seguenti

```
#Regola base per rilevare una scansione SYN di Nmap
alert tcp any any -> $HOME_NET any (msg:"Nmap SYN scan detected";
  flags:S; threshold: type both, track by_src, count 20, seconds 3;
  classtype:attempted-recon; sid:100001; rev:1;)
```

```
#Regola per rilevare uno scan TCP Connect
alert tcp any any -> $HOME_NET any (msg:"Nmap TCP Connect scan
  detected"; flags:S,A; threshold: type both, track by_src, count
  20, seconds 3; classtype:attempted-recon; sid:100002; rev:1;)
```

```
# Rule to detect TCP SYN scan di Emerging Threats(Nmap)
alert tcp any any -> $HOME_NET any (msg:"ET SCAN Nmap TCP Syn Scan";
  flags:S; threshold: type both, track by_src, count 5, seconds 60;
  reference:url,doc.emergingthreats.net/bin/view/Main/2000011;
  classtype:attempted-recon; sid:2000011; rev:13;)
```


Remediation

Se si identifica un IP sospetto che effettua scansioni, è possibile bloccare il traffico da quell'IP con una regola di iptables. In questo caso quindi si può eseguire il comando seguente per bloccare il traffico proveniente dal suddetto IP sospetto.

```
iptables -A INPUT -s 172.19.0.3 -j DROP
```

La soluzione più tempestiva consiste nel configurare Suricata in modalità IPS così che possa rilevare scansioni tramite regole specifiche e automaticamente bloccare le richieste sospette, inoltre è una soluzione più resistente se non sono noti gli indirizzi IP degli attaccanti a priori (come solitamente succede, talvolta con IP falsificati).

Nella modalità IPS, Suricata non si limita a rilevare, ma può anche bloccare il traffico in base alle regole configurate.

Per attuare questa soluzione sono richieste modifiche al file `suricata.yaml`, bisogna aggiungere una regola di tipo drop al file `suricata.rules` o aggiungere un nuovo file in cui tali regole sono contenute, come `emerging-scan.rules`, alla lista di rules files dopo aver abilitato le regole di drop, ed infine lanciare normalmente Suricata.

Di seguito una soluzione possibile con l'utilizzo di Suricata in modalità IPS:

Come documentato nella Documentazione di Suricata per avere suricata in modalità IPS tra le interfacce `eth0` ed `eth1` serve la seguente configurazione di `af-packet`:

modifica alla sezione af-packet in `/etc/suricata/suricata.yaml`

```
...  
  
af-packet:  
  - interface: eth0  
    threads: 1  
    defrag: yes  
    cluster-type: cluster_flow  
    cluster-id: 98  
    copy-mode: ips  
    copy-iface: eth1  
    buffer-size: 64535  
    use-mmap: yes  
  - interface: eth1  
    threads: 1  
    cluster-id: 97  
    defrag: yes  
    cluster-type: cluster_flow  
    copy-mode: ips  
    copy-iface: eth0  
    buffer-size: 64535  
    use-mmap: yes  
  
...
```

cluster-type: cluster_flow indica che il traffico verrà bilanciato in base ai flussi di dati, verranno raggruppati i pacchetti appartenenti allo stesso flusso e si assegnerà quel flusso a un singolo thread.

cluster-id identifica in modo univoco il cluster di pacchetti elaborati su ciascuna interfaccia.

In entrambe le interfacce viene specificato il **copy-mode: ips**. Ciò significa che Suricata opererà in modalità inline per bloccare i pacchetti.

In oltre con il parametro **copy-iface** (rispettivamente `copy-iface: eth1` e `copy-iface: eth0` per le interfacce `eth0` ed `eth1`) viene specificato che se un pacchetto viene bloccato allora non verrà inoltrato, creando così una protezione IPS tra le due interfacce.

Il parametro **defrag: yes** abilita la deframmentazione, utile per riassemblare pacchetti frammentati, ma può essere disattivata se il traffico non richiede tale operazione, ad esempio se si sta trattando principalmente con traffico non frammentato o se si vuole ridurre l'overhead.

La variabile **buffer-size: 64535** è la dimensione del buffer dei pacchetti per ciascuna interfaccia. Il valore 64535 è comunemente usato per bilanciare performance e utilizzo della memoria.

L'opzione **use-mmap: yes** permette a Suricata di utilizzare la memoria mappata per l'acquisizione dei pacchetti, migliorando le prestazioni nel gestire grandi volumi di traffico.

`/etc/suricata/rules/suricata.rules` (regola drop all'identificazione di uno scan)

```
drop tcp any any -> $HOME_NET any (msg:"Nmap TCP Connect scan
    detected"; flags:S,A; threshold: type both, track by_src, count
    20, seconds 3; classtype:attempted-recon; sid:100020; rev:1;)
```

Prima di avviare Suricata, si può verificare la validità del file di configurazione con il seguente comando per assicurarsi che non ci siano errori:

```
suricata -T -c /etc/suricata/suricata.yaml
```

e poi avviare suricata con il solito comando

```
suricata -c /etc/suricata/suricata.yaml -i eth0 -i eth1
```

3.1.2 XSS

Gli attacchi XSS (Cross-Site Scripting) sfruttano vulnerabilità nei siti web per iniettare script dannosi che vengono eseguiti nel browser degli utenti, consentendo il furto di dati o la compromissione delle sessioni.

Implementazione ed Identificazione attacco

XSS sfruttando la pagina `http://172.18.0.3/xssVuln.php` o `http://172.18.0.3/xssVulnCookie.php`. Per farlo da shell si puo' usare il comando `curl` come segue

```
curl 'http://172.18.0.3/xssVuln.php?comment=%3Cscript%3Ealert%28%27
XSS%21%27%29%3C%2Fscript%3E'
```

```
root@ef4920d9d01f:/# curl 'http://172.18.0.3/xssVuln.php?comment=%3Cscript%3Ealert%28%27XSS%21%27%29%3C%2Fscript%3E'
<!DOCTYPE html>
<html lang="it">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>XSS Vulnerable Website</title>
</head>
<body>
  <h1>Simulazione di un sito vulnerabile a XSS (Reflected)</h1>

  <p>Inserisci un commento:</p>
  <form action="index.html" method="GET">
    <label for="comment">Commento:</label><br>
    <textarea id="comment" name="comment" rows="4" cols="50"></textarea><br>
    <input type="submit" value="Invia">
  </form>

  <hr>

  <h2>Commenti inseriti:</h2>

  <div>
    <!-- Qui viene visualizzato il commento inserito dall'utente, qualsiasi
    input dell'utente, incluso codice JavaScript malevolo, verrà eseguito -->
    <p><script>alert('XSS!')</script></p>
  </div>
</body>
</html>
```

Figure 3.3: AttaccoXSS

```
{
  "timestamp": "2024-09-18T16:05:44.266110+0000",
  "flow_id": 246038394153840,
  "ln_iface": "eth0",
  "event_type": "alert",
  "src_ip": "172.19.0.3",
  "src_port": 51318,
  "dest_ip": "172.18.0.3",
  "dest_port": 80,
  "proto": "TCP",
  "pkt_src": "wire/pcap",
  "tx_id": 0,
  "alert": {
    "action": "allowed",
    "gid": 1,
    "signature_id": 1001001,
    "rev": 1,
    "signature": "XSS Attack Detected: Basic script tag",
    "category": "Web Application Attack",
    "severity": 1
  },
  "http": {
    "hostname": "172.18.0.3",
    "url": "/xssVuln.php?comment=%3Cscript%3Ealert%28%27XSS%21%27%29%3C%2Fscript%3E",
    "http_user_agent": "curl/7.50.1",
    "http_content_type": "text/html",
    "http_method": "GET",
    "protocol": "HTTP/1.1",
    "status": 200,
    "length": 820
  },
  "app_proto": "http",
  "direction": "to_server",
  "flow": {
    "pkts_to_server": 4,
    "pkts_to_client": 3,
    "bytes_to_server": 416,
    "bytes_to_client": 1198,
    "start": "2024-09-18T16:05:44.253893+0000",
    "src_ip": "172.19.0.3",
    "dest_ip": "172.18.0.3",
    "src_port": 51318,
    "dest_port": 80
  }
}
```

Figure 3.4: LogXSS

Una possibile regola per rilevare un tale attacco è la seguente

```
alert http any any -> $HOME_NET any (msg:"XSS Attack Detected: Basic
  script tag"; flow:to_server,established; content:"<script>";
  http_uri; nocase; classtype:web-application-attack; sid:1000004;
  rev:1;)
```

Remediation

Gli attacchi XSS sfruttano la mancanza di validazione o sanificazione dell'input nelle pagine web per inserire codice malevolo che viene poi eseguito nel browser degli utenti. La remediation per XSS si concentra principalmente sulla corretta gestione degli input e output in una pagina web.

Per prevenire tali attacchi, è necessario intervenire direttamente sul codice dell'applicazione web.

Un modo semplice ed efficace per prevenire gli attacchi XSS è trasformare i caratteri speciali (<, >, ", ', &, ecc.) all'interno dell'input in entità HTML. In PHP può essere sfruttata a tal scopo la funzione **htmlspecialchars()**. Oppure si possono aggiungere controlli sull'input dell'utente per accettare solo dati di un formato prestabilito (ad esempio solo testo e spazi).

La pagina `xssVuln.php` può quindi essere riscritta come segue sfruttando la funzione `htmlspecialchars()` per sanificare gli user input.

```
...
<div>
  <p><?php
    if(isset($_GET['comment'])) {
      $comment = htmlspecialchars($_GET['comment'], ENT_QUOTES, 'UTF
        -8');
      echo $comment;
    }
  ?></p>
</div>
...
```

Una soluzione alternativa potrebbe sfruttare le RegEx per validare l'input ed imporre dei constraints relativi alla forma.

3.1.3 DoS

Un attacco DoS (Denial of Service) mira a rendere un servizio o un sistema inaccessibile, spesso ciò si ottiene sovraccaricandolo di traffico o richieste.

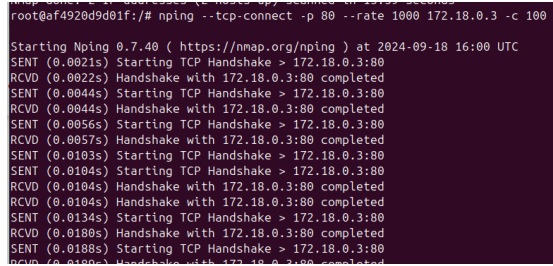
Implementazione ed Identificazione attacco

Attacco DoS, sfruttando ad esempio il comando nella suite `nmap`

CHAPTER 3. SVOLGIMENTO LABORATORIO (SOLUZIONE)

```
nping --tcp-connect -p 80 --rate 1000 172.18.0.3 -c 10000
```

dove `-c` in particolare specifica il numero di iterazioni dopo il quale fermarsi, il `rate` è il numero di pacchetti inviati al secondo, e ovviamente sono specificati IP destinazione, porta e tipo di connessione.



```
root@af4928d9d01f:/# nping --tcp-connect -p 80 --rate 1000 172.18.0.3 -c 100
Starting Nping 0.7.40 ( https://nmap.org/nping ) at 2024-09-18 16:00 UTC
SENT (0.0021s) Starting TCP Handshake > 172.18.0.3:80
RCVD (0.0022s) Handshake with 172.18.0.3:80 completed
SENT (0.0044s) Starting TCP Handshake > 172.18.0.3:80
RCVD (0.0044s) Handshake with 172.18.0.3:80 completed
SENT (0.0056s) Starting TCP Handshake > 172.18.0.3:80
RCVD (0.0057s) Handshake with 172.18.0.3:80 completed
SENT (0.0103s) Starting TCP Handshake > 172.18.0.3:80
SENT (0.0104s) Starting TCP Handshake > 172.18.0.3:80
RCVD (0.0104s) Handshake with 172.18.0.3:80 completed
RCVD (0.0104s) Handshake with 172.18.0.3:80 completed
SENT (0.0134s) Starting TCP Handshake > 172.18.0.3:80
RCVD (0.0188s) Handshake with 172.18.0.3:80 completed
SENT (0.0188s) Starting TCP Handshake > 172.18.0.3:80
RCVD (0.0189s) Handshake with 172.18.0.3:80 completed
```

Figure 3.5: AttaccoDoS



```
{
  "timestamp": "2024-09-18T16:00:19.979314+0000",
  "flow_id": 1109900978427521,
  "in_iface": "eth0",
  "event_type": "alert",
  "src_ip": "172.19.0.3",
  "src_port": 35077,
  "dest_ip": "172.18.0.3",
  "dest_port": 80,
  "proto": "TCP",
  "pkt_src": "wire/pcap",
  "alert": {
    "action": "allowed",
    "gid": 1,
    "signature_id": 100003,
    "rev": 1,
    "signature": "SYN Flood Detected",
    "category": "Attempted Denial of Service",
    "severity": 2
  },
  "direction": "to_server",
  "flow": {
    "pkts_toserver": 1,
    "pkts_toclient": 0,
    "bytes_toserver": 74,
    "bytes_toclient": 0,
    "start": "2024-09-18T16:00:19.979314+0000",
    "src_ip": "172.19.0.3",
    "dest_ip": "172.18.0.3",
    "src_port": 35077,
    "dest_port": 80
  }
}
```

Figure 3.6: LogDoS

La regola che si può usare per rilevare un tale attacco è la seguente

```
#Regola Suricata per rilevare l'attacco SYN flood (attacco DoS)
alert tcp any any -> $HOME_NET 80 (msg:"SYN Flood Detected"; flags:S;
    threshold: type both, track by_src, count 200, seconds 1;
    classtype:attempted-dos; sid:1000005; rev:1;)
```

Remediation

Analogamente a quanto succede per l'attacco scan, se si identifica un IP sospetto che genera un traffico anomalo, è possibile bloccare tale traffico proveniente da quell'IP con una regola di iptables. In questo caso quindi si può eseguire il comando seguente per bloccare il traffico proveniente dal suddetto ip sospetto.

```
iptables -A INPUT -s 172.19.0.3 -j DROP
```

Oppure tramite l'azione di Suricata in modalità IPS, come visto per la remediation per gli scan, aggiungendo la regola

```
drop tcp any any -> $HOME_NET 80 (msg:"SYN Flood Detected"; flags:S;
    threshold: type both, track by_src, count 200, seconds 1;
    classtype:attempted-dos; sid:1000005; rev:1;)
```

Altro metodo con iptables per evitare che il traffico anomalo generi un sovraccarico del servizio, consiste nel limitare il numero di connessioni HTTP (ossia sulla porta 80) ad esempio a 25 al minuto.

```
iptables -A INPUT -p tcp --dport 80 -m limit --limit 25/minute --
    limit-burst 100 -j ACCEPT
```

Entrambe però non risolvono del tutto il problema della availability se l'attacco DoS è distribuito.

Oppure, come visto nel paragrafo di remediation per gli scan, tramite l'azione di Suricata in modalità IPS (con configurazione analoga a quella del paragrafo Attacchi->Scan->Remediation) aggiungendo la regola

```
drop tcp any any -> $HOME_NET 80 (msg:"SYN Flood Detected"; flags:S;
    threshold: type both, track by_src, count 200, seconds 1;
    classtype:attempted-dos; sid:1000005; rev:1;)
```

3.1.4 SQL Injection

L'SQL Injection manipola le query SQL inviate a un database, permettendo a un attaccante di eseguire comandi arbitrari, come accedere o modificare dati sensibili.

Implementazione ed Identificazione attacco

L'implementazione dell'attacco Sql injection sfrutta la pagina <http://172.18.0.3/sqlVuln.php>, dove è presente un form da compilare per poter accedere ad un database. Per farlo da shell si può usare il comando curl come segue

```
curl http://172.18.0.3/sqlVuln.php?user_id=1'OR+1%3D1+%23
```

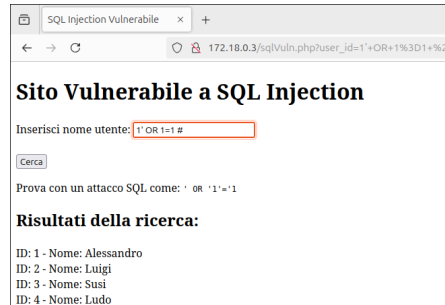


Figure 3.7: Attacco SQL Injection con browser

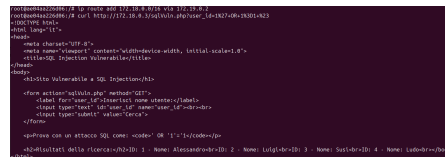


Figure 3.8: Attacco SQL Injection con curl

Ovviamente prima dell'esecuzione dell'attacco va innanzitutto creata e riempita la tabella del database come specificato nel paragrafo 2.1.2 sotto la voce Database MySQL.

Una regola che può rilevare un tale attacco è la seguente

```
alert http any any -> $HOME_NET any (msg:"SQL Injection Attempt - OR
1=1 URL-encoded"; content:"OR+1%3D1"; nocase; classtype:web-
application-attack; sid:1000011; rev:1;)
```

Tale regola in particolare specifica la codifica del pattern "OR 1=1" che viene effettuata da un browser quando viene inserita nell'URL della richiesta GET.

Però è importante sottolineare che tale regola è specifica per questo particolare attacco, tuttavia non è esaustiva per l'identificazione di tutti gli attacchi di questo tipo. Per aumentare la copertura servirebbero altre regole per aumentare il numero di query malevole che è possibile rilevare. Altre regole di questo tipo possono ad esempio essere le seguenti:

```
alert http any any -> $HOME_NET any (msg:"Possible SQL Injection
Attempt"; content:"union select"; nocase; content:"from
information_schema.tables"; nocase; classtype:web-application-
attack; sid:1000007; rev:1;)
```

```

alert http any any -> $HOME_NET any (msg:"Possible SQL Injection
  Attempt"; content:"select * from"; nocase; pcre:"/select[\s
  ]+[*]+[\s]+from[\s]+[\w]+/i"; classtype:web-application-attack;
  sid:1000008; rev:1;)

alert http any any -> $HOME_NET any (msg:"SQL Injection Attempt - OR
  1=1"; content:"OR 1=1"; nocase; classtype:web-application-attack;
  sid:1000009; rev:1;)

```

```

{
  "timestamp": "2024-09-20T12:19:26.833780+0000",
  "flow_id": 1883204066313981,
  "in_iface": "eth0",
  "event_type": "alert",
  "src_ip": "172.19.0.3",
  "src_port": 49960,
  "dest_ip": "172.18.0.3",
  "dest_port": 80,
  "proto": "TCP",
  "pkt_src": "wire/pcap",
  "tx_id": 0,
  "alert": {
    "action": "allowed",
    "gid": 1,
    "signature_id": 1000011,
    "rev": 1,
    "signature": "SQL Injection Attempt - OR 1=1",
    "category": "Web Application Attack",
    "severity": 1
  },
  "http": {
    "hostname": "172.18.0.3",
    "url": "/sqlVuln.php?user_id=1%27+OR+1%3D1+%23",
    "http_user_agent": "curl/7.50.1",
    "http_content_type": "text/html",
    "http_method": "GET",
    "protocol": "HTTP/1.1",
    "status": 200,
    "length": 692
  },
  "app_proto": "http",
  "direction": "to_server",
  "flow": {
    "pkts_toserver": 4,
    "pkts_toclient": 3,
    "bytes_toserver": 383,
    "bytes_toclient": 1070,
    "start": "2024-09-20T12:19:26.831683+0000",
    "src_ip": "172.19.0.3",
    "dest_ip": "172.18.0.3",
    "src_port": 49960,
    "dest_port": 80
  }
}

```

Figure 3.9: Log Sql Injection

Remediation

Separando l'input dalla query e aggiungendo una validazione dell'input, si può migliorare la struttura del codice e proteggere la pagina da SQL injection. Si possono quindi sfruttare i **Prepared Statements**, ossia delle query parametrizzate in cui si separano gli user input da una query con struttura fissa usata per eseguire la stessa interrogazione ripetitivamente con alta efficienza e soprattutto permette la protezione dalle SQL injections.

La pagina sqlVuln.php quindi può essere riscrivere come segue sfruttando un prepared statement.

...

```
<?php
// Creazione della connessione
$servername = "10.0.0.4"; // IP del container del database
$username = "user";
$password = "password";
$dbname = "testdb";
$conn = new mysqli($servername, $username, $password, $dbname);

// Controllo della connessione
if ($conn->connect_error) {
    die("Connessione al database fallita: " . $conn->connect_error);
}

// Funzione di validazione dell'input
function validaInput($data) {
    // Controlliamo se il campo è vuoto
    if (empty($data)) {
        return false;
    }
    // Rimuoviamo eventuali spazi bianchi
    $data = trim($data);
    // Filtriamo caratteri non desiderati con una RegEx (lettere e spazi)
    if (!preg_match("/^[a-zA-Z ]*$/", $data)) {
        return false;
    }
    return $data;
}

// Verifica se è stato inviato un parametro tramite GET
if (isset($_GET['user_id'])) {
    $user = validaInput($_GET['user_id']);

    if ($user === false) {
        echo "Input non valido. Solo caratteri alfabetici";
    } else {
        // Preparazione della query sicura con prepared statements
        $stmt = $conn->prepare("SELECT * FROM utenti WHERE nome = ?");
        $stmt->bind_param("s", $user);
    }
}
```

```

$stmt->execute();
$result = $stmt->get_result();

// Controllo del risultato
if ($result->num_rows > 0) {
    // Output dei dati di ciascuna riga
    while ($row = $result->fetch_assoc()) {
        echo "ID: " . $row["id"] . " - Nome: " . $row["nome"] . "<br>";
    }
} else {
    echo "0 risultati trovati";
}

// Chiusura dello statement
$stmt->close();
}
}

// Chiusura della connessione
$conn->close();
?>
...

```

Come si può notare dal codice la query **"SELECT * FROM utenti WHERE nome = ?"** presenta un parametro (ossia il '?') che è affidato alla variabile **user**, la quale è ottenuta dal parametro **user_id** inserito dall'utente ed inglobato nell'URL.

Tale parametro prima di essere unito (attraverso la funzione **bind_param()**) alla query viene prima passato alla funzione **validaInput()** che controlla che il parametro in questione rispetti la forma specificata attraverso la **Regular Expression** (Regex) **"/[a-zA-Z]*\$/"** che impone l'uso di sole lettere minuscole, maiuscole oppure spazi senza vincoli sulla dimensione.

Infine, dopo aver legato il parametro alla struttura predeterminata con **bind_param()**, la query viene inoltrata al database tramite l'utilizzo del metodo **execute()**.