



UNIVERSITÀ DI PERUGIA
Dipartimento di Matematica e Informatica



CYBERSECURITY

ELK docker server

Professore

Prof. Bistarelli Stefano
Prof. Santini Francesco

Studente

Bocchini Fabio mat. 340600

Anno Accademico 2022-2023

Indice

1	Introduzione	4
1.1	Obiettivo	4
1.2	Strumenti	4
1.2.1	Docker	4
1.2.2	Docker Compose	5
1.2.3	Elasticsearch	5
1.2.4	Logstash	5
1.2.5	Kibana	5
1.2.6	Filebeat	5
1.2.7	Iptables	6
2	Configurazioni	7
2.1	Generali	7
2.1.1	cyebr.sh	7
2.1.2	.env	9
2.1.3	setup/instances.yml	10
2.1.4	setup/setup.sh	11
2.2	Docker	14
2.2.1	docker-compose.yml	14
2.2.2	docker-compose.setup.yml	19
2.3	Elasticsearch	20
2.3.1	elastichsearch.yml	20
2.4	Logstash	21
2.4.1	logstash.yml	21
2.4.2	pipeline/logstash.conf	22
2.4.3	pipeline/patterns/iptables.pattern	24
2.5	Kibana	26
2.5.1	kibana.yml	26
2.6	Filebeat	27
2.6.1	filebeat.yml	27
2.6.2	iptable-rules.sh	28

3	Monitoraggio	29
3.1	Avvio	29
3.2	Kibana	29
4	Analisi dei Dati	32
4.1	Port Scan Attack	32
4.2	L'attacco	32
4.3	Regole	34
4.4	Test	36
4.5	Altre Regole	38
5	Fonti	39

1 Introduzione

1.1 Obiettivo

L'obiettivo di questo progetto è quello di simulare tramite **Docker** due server **Linux** che utilizzano i servizi dello **stack ELK** per collezionare e gestire file di log in stile **server SIEM**. Più nello specifico i due server saranno:

1. Il primo conterrà i servizi principali dello **stack ELK**, ovvero **Elasticsearch**, **Logstash** e **Kibana**, assumerà quindi la funzione di un **server SIEM (Security Information and Event Management)**, ovvero un server che in modo centralizzato **analizza** e **monitora** i dati ricevuti da sonde esterne con l'obiettivo di **individuare vulnerabilità** nel sistema.
2. Il secondo invece simulerà un secondo server nella rete, contenente un **firewall** e una **sonda Filebeat** (parte della famiglia **Beats**). Questa sonda **colleziona** i dati e li invia al primo server per essere elaborati e analizzati.

1.2 Strumenti

1.2.1 Docker

Docker è uno strumento che facilita la **creazione**, il **deployment** e l'**esecuzione** di applicazioni attraverso l'utilizzo di **container**. I container permettono di racchiudere un'applicazione insieme a tutte le sue parti, come le **librerie** e le altre **dipendenze**, e di distribuirla come un unico pacchetto. Utilizzando i container, possiamo essere sicuri che l'applicazione funzionerà su qualsiasi altra macchina indipendentemente dalle differenze di sistema operativo, software o configurazioni con la macchina utilizzata per lo sviluppo.

Docker permette di avviare le applicazioni senza dover installare o configurare alcuna infrastruttura. Inoltre, rende facile eseguire più applicazioni sulla stessa macchina, poiché ogni applicazione può essere eseguita, avviata o fermata in un proprio container ed essere isolata da tutte le altre.

1.2.2 Docker Compose

Docker Compose è uno strumento per la definizione e l'esecuzione di applicazioni **multi-container** utilizzando il file di configurazione Docker Compose. Permette quindi di creare e gestire facilmente un gruppo di container Docker (ognuno con i suoi volumi, porte, variabili d'ambiente ecc.), che lavorano insieme per eseguire un'applicazione.

1.2.3 Elasticsearch

Elasticsearch è un **motore di ricerca** open source basato sull'indice **Lucene**. Viene spesso utilizzato per la ricerca **full-text**, la ricerca geografica e l'**analisi dei dati in tempo reale**. In questo caso viene utilizzato per la ricerca full-text dei file di log di Iptables.

1.2.4 Logstash

Logstash è uno strumento open source per la **raccolta**, l'**elaborazione** e il **trasferimento** dei dati di log. Può raccogliere dati da una vasta gamma di fonti, come file di log, eventi system e dati di rete, e quindi elaborarli utilizzando plugin per **trasformare** e **modificare** i dati. Una volta elaborati, i dati possono essere trasmessi ad un output finale, come Elasticsearch.

1.2.5 Kibana

Kibana è l'applicazione **front-end web**, quindi visualizzabile nel browser, utilizzata per **visualizzare** ed **utilizzare** i dati elaborati da Elasticsearch.

1.2.6 Filebeat

Filebeat è uno strumento per l'invio di dati da macchine esterne a server Logstash o Elasticsearch. Fa parte della famiglia **Beats**, un insieme di **sonde** che raccolgono dati di diverso tipo, come **metric data** per il monitoraggio del sistema, **dati di rete** per analizzare i dati di specifici protocolli, Beats specifici per **dati cloud** e molti altri.

Filebeat è il Beat utilizzato per la spedizione di **file di log generici**.

1.2.7 Iptables

Iptables è un **firewall** per linux implementato a livello **kernel** e configurabile da linea di comando. Permette la creazione di **catene di regole** per la gestione granulare dei pacchetti che entrano, escono e passano dalla macchina.

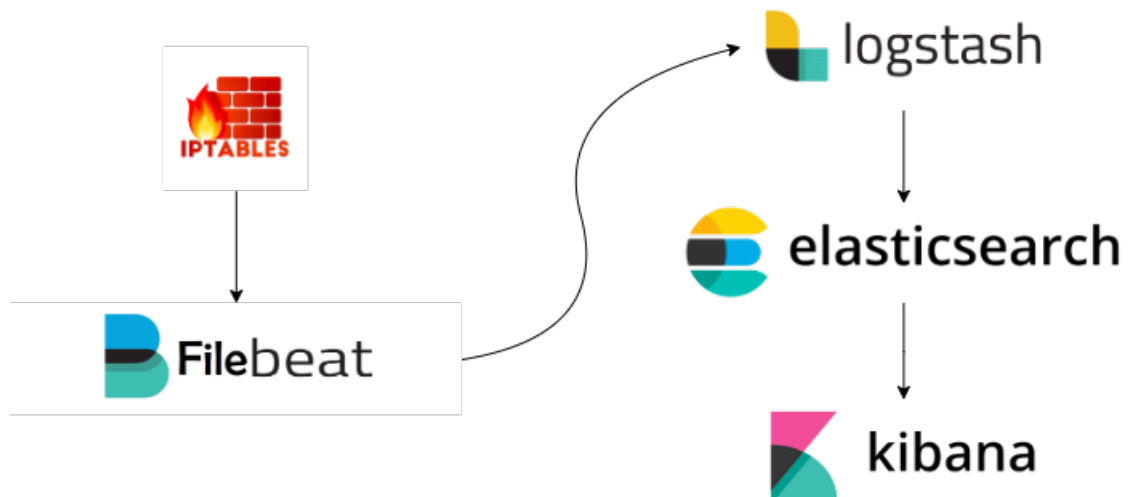


Figura 1: Esempio del flusso dati

2 Configurazioni

Di seguito vediamo le configurazioni utilizzate per i vari software

2.1 Generali

2.1.1 cyebr.sh

```
#!/usr/bin/env /bin/bash

Green="\033[1;92m"    # Green
Reset="\033[0m"        # Text Reset

PROJECT_HOME=$(cd -P -- "$(dirname -- "$0")" && pwd -P)
DOCKER_COMPOSE="${PROJECT_HOME}/docker-compose.yml"
DOCKER_COMPOSE_SETUP="${PROJECT_HOME}/docker-compose.setup.yml"

while [[ $# -gt 0 ]]
do
    key="$1"

    case "$key" in
        setup)
            docker-compose -f "$DOCKER_COMPOSE_SETUP" up --build
            ;;
        up)
            docker-compose -f "$DOCKER_COMPOSE" up -d --build --remove-orphans
            docker exec -it filebeat sh -c "ulogd -d"
            docker exec --privileged -it filebeat /home/iptables-rules.sh
            exit
            ;;
        down)
            docker-compose -f "$DOCKER_COMPOSE" down --volumes
            exit
            ;;
        clean)
            rm -r $PROJECT_HOME/secrets/*
            exit
            ;;
    esac

    shift
done

echo -en ""
Usage:
- ${Green}setup${Reset}: creazione dei certificati di sicurezza
- ${Green}up${Reset}: start dei servizi
- ${Green}down${Reset}: stop sei servizi docker
- ${Green}clean${Reset}: rimuove tutti i file generati da setup
""
```

Figura 2: cyber.sh

Il file **cyber.sh** è uno script che permette di gestire il progetto con comandi semplici. Può essere utilizzato nel seguente modo:

- **./cyber.sh setup**: Esegue lo script presente nella cartella setup per la generazione del keystore e dei certificati TLS che verranno utilizzati dai vari servizi.
- **./cyber.sh up**: Esegue il comando `docker-compose up` per avviare tutti i container necessari. Viene poi avviato il daemon ulogd e vengono applicate le regole di iptables.
- **./cyber.sh down**: Ferma e rimuove i container relativi al progetto in esecuzione.
- **./cyber.sh clean**: Elimina i file generati da `./cyber.sh setup`.

Per utilizzare il file è necessario renderlo eseguibile dal proprio utente con il comando

```
sudo chmod +x cyber.sh
```


2.1.2 .env

```
ELK_VERSION=7.16.0

# elastic
ELASTIC_USERNAME=elastic
ELASTIC_PASSWORD=pwdpwd

# kibana
KIBANA_URL=https://kibana:5601

# memoria
ELASTICSEARCH_HEAP=2g
LOGSTASH_HEAP=1g
FILEBEAT_HEAP=256m
XPACK_ENCRYPTION_KEY=somesuperlongstringlikethisoneMQBbtsynu4bV2uxLy

# certificati tls
CA_PASSWORD=pwdpwd
CA_DAYS=3650

# percorsi
ELASTIC_DIR=/usr/share/elasticsearch
LOGSTASH_DIR=/usr/share/logstash
KIBANA_DIR=/usr/share/kibana
FILEBEAT_DIR=/etc/filebeat

# self-signed certificates
STAGING=true
```

Figura 3: .env

Il file .env definisce le variabili d'ambiente che verranno utilizzate dai file docker-compose. Viene riconosciuto automaticamente, quindi non c'è bisogno di specificare quale file env debba essere utilizzato.

Qui definiamo:

- la versione dei vari software ELK da utilizzare.
- le credenziali elastic per autenticarci su kibana.
- i limiti di memoria per i servizi
- le impostazioni per la generazione dei certificati
- i percorsi dove trovare le configurazioni per i vari servizi

2.1.3 setup/instances.yml

```
instances:
- name: elasticsearch
  dns:
    - elasticsearch
    - localhost
  ip:
    - 0.0.0.0
    - 127.0.0.1

- name: kibana
  dns:
    - kibana
    - localhost
  ip:
    - 0.0.0.0
    - 127.0.0.1

- name: logstash
  dns:
    - logstash
    - localhost
  ip:
    - 0.0.0.0
    - 127.0.0.1

- name: filebeat
  dns:
    - filebeat
    - localhost
  ip:
    - 0.0.0.0
    - 127.0.0.1
```

Figura 4: setup/instances.yml

In questo file definiamo i vari servizi con relativi hostname che avranno bisogno di un certificato ssl. Verrà utilizzato da setup.sh.

2.1.4 setup/setup.sh

Questo file utilizza il servizio elasticsearch per generare il keystore e i certificati **SSL (Secure Socket Layer)** per ogni servizio che li necessita. Verrà lanciato all'interno di un container docker, quindi all'inizio installa i pacchetti necessari, rimuove i certificati se già ce ne sono all'interno della cartella, poi genera i certificati per ogni servizio definito in `instances.yml`.

Il protocollo **TLS (Transport Layer Security)** è un protocollo crittografico che permette una comunicazione sicura su reti TCP/IP fornendo autenticazione, integrità dei dati e confidenzialità. In questo caso viene utilizzato per la comunicazione tra in vari servizi dello stack ELK.

Il protocollo **HTTPS (HyperText Transfer Protocol over Secure Socket Layer)** viene utilizzato per lo stesso scopo, ma in questo caso le comunicazioni crittate solo solamente quelle tra il servizio Kibana e l'utente che interagisce tramite browser.

```

# Exit on Error
set -e

CONFIG_DIR=/usr/share/elasticsearch/config
OUTPUT_FILE=/secrets/elasticsearch.keystore
NATIVE_FILE=$CONFIG_DIR/elasticsearch.keystore
OUTPUT_DIR=/secrets
CA_DIR=$OUTPUT_DIR/certificate_authority
KEYSTORES_DIR=$OUTPUT_DIR/keystores
CERT_DIR=$OUTPUT_DIR/certificates
CA_P12=$CA_DIR/elastic-stack-ca.p12
CA_ZIP=$CA_DIR/ca.zip
CA_CERT=$CA_DIR/ca/ca.crt
CA_KEY=$CA_DIR/ca/ca.key
BUNDLE_ZIP=$OUTPUT_DIR/bundle.zip
CERT_KEYSTORES_ZIP=$OUTPUT_DIR/cert_keystores.zip
HTTP_ZIP=$OUTPUT_DIR/http.zip

apt-get update
apt-get install unzip openssl -y

create_self_signed_ca()
{
    printf "==== Creating Self-Signed Certificate Authority =====\n"
    printf "===== \n"
    echo "Generating Self-Signed Certificate Authority PEM ..."
    bin/elasticsearch-certutil ca --pass "" --pem --out $CA_ZIP --silent
    unzip $CA_ZIP -d $CA_DIR
    echo "Generating Self-Signed Certificate Authority P12 ..."
    bin/elasticsearch-certutil ca --pass "" --out $CA_P12 --silent
    echo "elastic-stack-ca.p12 is located $CA_P12"
}

create_certificates()
{
    printf "==== Generating Certificate Keystores =====\n"
    printf "===== \n"
    echo "Creating p12 certificate keystores"
    bin/elasticsearch-certutil cert --silent --in $CONFIG_DIR/instances.yml --out
    $CERT_KEYSTORES_ZIP --ca $CA_P12 --ca-pass "" --pass ""
    unzip $CERT_KEYSTORES_ZIP -d $KEYSTORES_DIR
    echo "Creating crt and key certificates"
    bin/elasticsearch-certutil cert --silent --in $CONFIG_DIR/instances.yml --out
    $BUNDLE_ZIP --ca-cert $CA_CERT --ca-key $CA_KEY --ca-pass "" --pem
    unzip $BUNDLE_ZIP -d $CERT_DIR
}

setup_passwords()
{
    printf "==== Setting up Default User Passwords =====\n"
    printf "===== \n"

    bin/elasticsearch-setup-passwords auto -u "https://0.0.0.0:9200" -v --batch
}

create_keystore()
{
    printf "==== Creating Elasticsearch Keystore =====\n"
    printf "===== \n"
    elasticsearch-keystore create >> /dev/null
}

```

```

## Setting Bootstrap Password
echo "Setting bootstrap password..."
(echo "$ELASTIC_PASSWORD" | elasticsearch-keystore add -x 'bootstrap.password')

# Replace current Keystore
if [ -f "$OUTPUT_FILE" ]; then
    echo "Remove old elasticsearch.keystore"
    rm $OUTPUT_FILE
fi

#setup_passwords
echo "Saving new elasticsearch.keystore"
mv $NATIVE_FILE $OUTPUT_FILE
chmod 0644 $OUTPUT_FILE

printf "===== Keystore setup completed successfully =====\n"
printf "=====\n"
}

remove_existing_certificates()
{
    printf "===== Removing Existing Secrets =====\n"
    printf "=====\n"
    for f in $OUTPUT_DIR/* ; do
        if [ -d "$f" ]; then
            echo "Removing directory $f"
            rm -rf $f
        fi
        if [ -f "$f" ]; then
            echo "Removing file $f"
            rm $f
        fi
    done
}

create_directory_structure()
{
    printf "===== Creating Required Directories =====\n"
    printf "=====\n"
    echo "Creating Certificate Authority Directory..."
    mkdir $CA_DIR
    echo "Creating Keystores Directory..."
    mkdir $KEYSTORES_DIR
    echo "Creating Certificates Directory..."
    mkdir $CERT_DIR
}

remove_existing_certificates
create_directory_structure
create_keystore
create_self_signed_ca
create_certificates

openssl pkcs8 -in /secrets/certificates/logstash/logstash.key -topk8 -nocrypt -out
/secrets/certificates/logstash/logstash.pkcs8.key

chown -R 1000:0 $OUTPUT_DIR

printf "=====\n"
printf "SSL Certificates generation completed successfully.\n"
printf "=====\n"

```

Figura 5: setup/setup.sh

2.2 Docker

2.2.1 docker-compose.yml

```
version: '3.8'

volumes:
  data:

secrets:
  ca.crt:
    file: ./secrets/certificate_authority/ca/ca.crt
    [...]

services:
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:${ELK_VERSION}
    container_name: elasticsearch
    environment:
      CONFIG_DIR: ${ELASTIC_DIR}/config
      ELASTIC_USERNAME: ${ELASTIC_USERNAME}
      ELASTIC_PASSWORD: ${ELASTIC_PASSWORD}
      ES_JAVA_OPTS: -Xmx${ELASTICSEARCH_HEAP} -Xms${ELASTICSEARCH_HEAP}
      bootstrap.memory_lock: "true"
      discovery.type: single-node
    healthcheck:
      test: curl --fail elasticsearch:9200 || exit 1
      interval: 5s
      timeout: 10s
      retries: 100
    hostname: elasticsearch
    networks:
      - cyber-network
    ports:
      - 9200:9200
    restart: unless-stopped
    secrets:
      - source: elasticsearch.keystore
        target: ${ELASTIC_DIR}/config/elasticsearch.keystore
      - source: ca.crt
        target: ${ELASTIC_DIR}/config/ca.crt
      - source: elasticsearch.cert
        target: ${ELASTIC_DIR}/config/elasticsearch.crt
      - source: elasticsearch.key
        target: ${ELASTIC_DIR}/config/elasticsearch.key
    volumes:
      - data:${ELASTIC_DIR}
      -
    ./config/elasticsearch/elasticsearch.yml:${ELASTIC_DIR}/config/elasticsearch.yml:ro
```

```

kibana:
  image: docker.elastic.co/kibana/kibana:${ELK_VERSION}
  container_name: kibana
  depends_on:
    - elasticsearch
  environment:
    CONFIG_DIR: ${KIBANA_DIR}/config
    ELASTIC_USERNAME: ${ELASTIC_USERNAME}
    ELASTIC_PASSWORD: ${ELASTIC_PASSWORD}
    ENCRYPTION_KEY: ${XPACK_ENCRYPTION_KEY}
    KIBANA_URL: ${KIBANA_URL}
  healthcheck:
    test: curl --fail kibana:4601 || exit 1
    interval: 5s
    timeout: 10s
    retries: 100
  hostname: kibana
  networks:
    - cyber-network
  ports:
    - 5601:5601
  restart: unless-stopped
  secrets:
    - source: ca.crt
      target: ${KIBANA_DIR}/config/ca.crt
    - source: kibana.crt
      target: ${KIBANA_DIR}/config/kibana.crt
    - source: kibana.key
      target: ${KIBANA_DIR}/config/kibana.key
  volumes:
    - ./config/kibana/kibana.yml:${KIBANA_DIR}/config/kibana.yml:ro

logstash:
  image: docker.elastic.co/logstash/logstash:${ELK_VERSION}
  container_name: logstash
  depends_on:
    - elasticsearch
    - kibana
  environment:
    path.settings: null
    CONFIG_DIR: ${LOGSTASH_DIR}/config
    ELASTIC_USERNAME: ${ELASTIC_USERNAME}
    ELASTIC_PASSWORD: ${ELASTIC_PASSWORD}
    LS_JAVA_OPTS: "-Xmx${LOGSTASH_HEAP} -Xms${LOGSTASH_HEAP}"
  hostname: logstash
  networks:
    - cyber-network
  ports:
    - 5044:5044
    - 5045:5045
  restart: unless-stopped
  secrets:
    - source: ca.crt
      target: ${LOGSTASH_DIR}/config/ca.crt
    - source: logstash.crt
      target: ${LOGSTASH_DIR}/config/logstash.crt
    - source: logstash.pkcs8.key
      target: ${LOGSTASH_DIR}/config/logstash.pkcs8.key
    - source: logstash.key
      target: ${LOGSTASH_DIR}/config/logstash.key
    - source: logstash.p12
      target: ${LOGSTASH_DIR}/config/logstash.p12
  volumes:
    - ./config/logstash/logstash.yml:${LOGSTASH_DIR}/config/logstash.yml
    - ./config/logstash/pipeline/logstash.conf:${LOGSTASH_DIR}/pipeline/logstash.conf
    - ./config/logstash/pipeline/patterns/iptables.pattern:${LOGSTASH_DIR}/pipeline/patterns/iptables.pattern

```

```

filebeat:
  build: ./config/filebeat
  command: filebeat -e -d "publish" && tail -f /dev/null
  cap_add:
    - NET_ADMIN
  container_name: filebeat
  depends_on:
    - elasticsearch
    - kibana
  environment:
    CONFIG_DIR: ${FILEBEAT_DIR}/config
    LS_JAVA_OPTS: "-Xmx${FILEBEAT_HEAP} -Xms${FILEBEAT_HEAP}"
    ELASTIC_USERNAME: ${ELASTIC_USERNAME}
    ELASTIC_PASSWORD: ${ELASTIC_PASSWORD}
  hostname: filebeat
  networks:
    - cyber-network
  ports:
    - 21:21
    - 22:22
    - 80:80
    - 9000:9000
  secrets:
    - source: ca.crt
      target: ${FILEBEAT_DIR}/config/ca.crt
    - source: filebeat.crt
      target: ${FILEBEAT_DIR}/config/filebeat.crt
    - source: filebeat.key
      target: ${FILEBEAT_DIR}/config/filebeat.key
  volumes:
    - ./config/filebeat/filebeat.yml:${FILEBEAT_DIR}/filebeat.yml:ro

networks:
  cyber-network:
    driver: bridge

```

Figura 6: docker-compose.yml

Questo è il file docker-compose che permette di avviare e gestire tutti i container insieme. Sono specificati tutti i servizi che vogliamo lanciare, ognuno di loro ha le proprie configurazioni come:

- **image:** Specifica da quale immagine inizializzare il container. Per i tre servizi ELK vengono usate le immagini ufficiali consigliate dalla documentazione.
- **build:** Per il servizio Filebeat invece viene definito il path per il Dockerfile scritto a mano che vedremo nella sezione dedicata a Filebeat.
- **cap_add:** Si possono aggiungere o rimuovere Capabilities ai vari servizi. In questo caso aggiungiamo la Capability NET_ADMIN al container che utilizzerà iptables per permetterci di poter applicare regole al firewall.
- **container_name:** Il nome che verrà dato al container da Docker.
- **depends_on:** Specifica quali da quali altri servizi dipende il container, che quindi verrà avviato solo dopo che gli altri sono pronti.
- **environment:** Qui si possono popolare le variabili d'ambiente necessarie per ogni macchina.
- **healthcheck:** Definisce un test per controllare se il container è "healthy", ovvero sano e pronto all'esecuzione. Questo permette agli altri container dipendenti da questo di aspettare prima di essere avviati.
- **hostname:** Definisce un hostname per il container condiviso da tutti gli altri container, così da non dover utilizzare indirizzi IP per la comunicazione.
- **networks:** Definisce il network utilizzato dal container, nel nostro caso la rete "cyber-network" è definita in fondo al file.
- **ports:** Mappatura delle porte esposte all'esterno, vengono mappate solamente quelle utilizzate dai servizi che ci interessano nella forma porta_locale:porta_container.
- **secrets:** Vengono specificati quali file sono sensibili e quindi devono essere trattati come tali. Questi file infatti non verranno mandati in chiaro e il loro accesso sarà ristretto a chi non è autorizzato. Qui salviamo il keystore e i certificati necessari.

- **volumes** File e cartelle condivisi con la macchina locale. Nella forma volume_interno:volume_esterno:opzioni.

2.2.2 docker-compose.setup.yml

```
version: '3.8'

services:
  certs:
    container_name: certs
    image: docker.elastic.co/elasticsearch/elasticsearch:${ELK_VERSION}
    command: bash ${ELASTIC_DIR}/config/setup.sh
    user: "0"
    volumes:
      - ./secrets:/secrets/
      - ./setup/setup.sh:${ELASTIC_DIR}/config/setup.sh
      - ./setup/instances.yml:${ELASTIC_DIR}/config/instances.yml:ro
    environment:
      ELASTIC_PASSWORD: ${ELASTIC_PASSWORD}
      STAGING: ${STAGING}

volumes:
  secrets:
    driver: local
  setup:
    driver: local
```

Figura 7: docker-compose.setup.yml

Dato che per la creazione dei certificati abbiamo bisogno di elasticsearch, questo docker-file verrà utilizzato solo per questo. Una volta lanciato userà lo script `setup.sh` per generare i certificati in un volume condiviso, da dove poi l'altro docker-file li prenderà e utilizzerà per i container che dovranno rimanere in esecuzione.

2.3 Elasticsearch

I file relativi a logstash si trovano in `/docker/config/elasticsearch`

2.3.1 elastichsearch.yml

```
## Cluster Settings
cluster.name: "elk-tls-cluster"
network.host: "0.0.0.0"
http.host: 0.0.0.0

## License
xpack.license.self_generated.type: trial

# Security
xpack.security.enabled: true
xpack.security.authc.token.enabled: true
xpack.security.authc.api_key.enabled: true

# transport security settings
# This is mostly used for inter-node communications between parts of the ELK stack
xpack.security.transport.ssl.enabled: true
xpack.security.transport.ssl.key: ${CONFIG_DIR}/elasticsearch.key
xpack.security.transport.ssl.certificate: ${CONFIG_DIR}/elasticsearch.crt
xpack.security.transport.ssl.certificate_authorities: ${CONFIG_DIR}/ca.crt

# HTTP security settings
# This is used for client server ssl/tls communications (e.g. browser to kibana)
xpack.security.http.ssl.enabled: true
xpack.security.http.ssl.key: ${CONFIG_DIR}/elasticsearch.key
xpack.security.http.ssl.certificate: ${CONFIG_DIR}/elasticsearch.crt
xpack.security.http.ssl.certificate_authorities: ${CONFIG_DIR}/ca.crt
```

Figura 8: elastichsearch.yml

Qui troviamo il file di configurazione generale per elasticsearch. Vi sono definite, oltre al nome del cluster e gli host al quale deve connettersi, le impostazioni per l'utilizzo di protocolli di sicurezza tramite `xpack.security`.

Viene abilitata infatti l'autenticazione tramite **token** e tramite **chiave api**, poi viene abilitato l'ssl sia per le comunicazioni interne che per le comunicazioni tramite HTTP e specificati i vari path per i certificati.

2.4 Logstash

I file relativi a logstash si trovano in `/docker/config/logstash`

2.4.1 logstash.yml

```
node.name: "logstash"
http.host: "0.0.0.0"

path.settings: ${CONFIG_DIR}

# X-Pack Security Options
xpack.management.elasticsearch.username: "${ELASTIC_USERNAME}"
xpack.management.elasticsearch.password: "${ELASTIC_PASSWORD}"

xpack.monitoring.elasticsearch.username: "${ELASTIC_USERNAME}"
xpack.monitoring.elasticsearch.password: "${ELASTIC_PASSWORD}"
xpack.monitoring.elasticsearch.ssl.certificate_authority: "${CONFIG_DIR}/ca.crt"
```

Figura 9: logstash.yml

Questo è il file di configurazione generale per il servizio Logstash. Vengono passati i dati di sicurezza xpack tramite le variabili d'ambiente definite in `docker-compose.yml`

2.4.2 pipeline/logstash.conf

```
input {
  beats {
    port => 5044
    ssl => true
    ssl_certificate => "${CONFIG_DIR}/logstash.crt"
    ssl_key => "${CONFIG_DIR}/logstash.pkcs8.key"
  }
}
filter {
  grok {
    patterns_dir => ["/usr/share/logstash/pipeline/patterns"]
    match => { "message" => "%{IPTABLES}" }
  }
}
output {
  elasticsearch {
    hosts => ["https://elasticsearch:9200"]
    user => "${ELASTIC_USERNAME}"
    password => "${ELASTIC_PASSWORD}"
    ssl => true
    ssl_certificate_verification => true
    cacert => "${CONFIG_DIR}/ca.crt"
    index => "%{[@metadata][beat]}-%{[@metadata][version]}"
  }
}
```

Figura 10: pipeline/logstash.conf

Questo file definisce come logstash deve comportarsi. Viene definito **pipeline** e possono esserne presenti diverse, ognuna che gestisce input diversi.

Un file pipeline è diviso in 3 parti:

1. **input**: Qui viene definito dove logstash deve raccogliere i dati. In questo caso Filebeat glieli invierà sulla porta **5044** in modo criptato. Possono essere definiti altri tipi di input, come lo **stdin**, un **file** nel filesistem locale o anche altre **pipeline**, in modo da creare catene.
2. **filter**: Qui possiamo andare a manipolare i dati in entrata. Ci sono vari plugin disponibili, ma uno dei più utilizzati è **grok**, che permette di trasformare un testo in formato **json** tramite dei pattern che vengono riconosciuti. In questo caso il pattern è definito in un file e tutto ciò che viene riconosciuto su quel pattern viene strutturato in json.
3. **output**: Qui definiamo cosa logstash deve fare con i dati manipolati. L'opzione elasticsearch li invia al servizio con la possibilità di utilizzare protocolli ssl.

C'è anche la possibilità di scrivere l'output su **stdout**, di scriverli su un **file**, mandarli per **email** o molto altro.

2.4.3 pipeline/patterns/iptables.pattern

```
IPTABLES_INCOMPLETE_PACKET INCOMPLETE \[%{NUMBER:incomplete} bytes\]

IPTABLES_ICMP_EXTRA_ECHO ID=%{NUMBER:icmp_id} SEQ=%{NUMBER:icmp_seq}
IPTABLES_ICMP_EXTRA_PARAM PARAMETER=%{NUMBER:icmp_parameter}
IPTABLES_ICMP_EXTRA_REDIRECT GATEWAY=%{IP:icmp_redirect}
IPTABLES_ICMP_EXTRA ( (?:%{IPTABLES_ICMP_EXTRA_ECHO}|%{IPTABLES_ICMP_EXTRA_PARAM}|%
{IPTABLES_ICMP_EXTRA_REDIRECT}))*

IPTABLES_IP_FRAGFLAG ((?<= )(CE|DF|MF))*
IPTABLES_PROTOCOL PROTO=(?<proto>[a-zA-Z0-9]+)
IPTABLES_PORT_PAIR SPT=%{NUMBER:src_port} DPT=%{NUMBER:dst_port}

IPTABLES_TCP_FLAGS ((?<= )(CWR|ECE|URG|ACK|PSH|RST|SYN|FIN))*
IPTABLES_TCP_SEQ SEQ=%{NUMBER:seq_seq} ACK=%{NUMBER:seq_ack}
IPTABLES_TCP_DETAILS (?:%{IPTABLES_TCP_SEQ} )?WINDOW=%{NUMBER>window} RES=0x%
{BASE16NUM:res} %{IPTABLES_TCP_FLAGS:tcp_flags}
IPTABLES_UDP_DETAILS LEN=%{NUMBER:udp_len}

IPTABLES_ICMP_DETAILS TYPE=%{NUMBER:icmp_type} CODE=%{NUMBER:icmp_code}(( %
{IPTABLES_INCOMPLETE_PACKET})|%{IPTABLES_ICMP_EXTRA})

IPTABLES_IP SRC=%{IP:src_ip} DST=%{IP:dst_ip} LEN=%{NUMBER:length} TOS=%{BASE16NUM:tos}
PREC=0x%{BASE16NUM:prec} TTL=%{NUMBER:tll} ID=%{NUMBER:id}(?: %
{IPTABLES_IP_FRAGFLAG:fragment_flags})?(?: FRAG: %{NUMBER:fragment})?

IPTABLES_IP_PAYLOAD %{IPTABLES_PROTOCOL}( %{IPTABLES_PORT_PAIR})?( (%
{IPTABLES_TCP_DETAILS}|%{IPTABLES_UDP_DETAILS}|%{IPTABLES_ICMP_DETAILS})|%
{IPTABLES_INCOMPLETE_PACKET}))?

PREFIX %{SYSLOGTIMESTAMP:timestamp} %{SYSLOGHOST:logsource} %{SYSLOGHOST:direction}:
IPTABLES_ETHERNET IN=%{DATA:in_device} OUT=%{DATA:out_device} MAC=%{DATA:mac}

IPTABLES %{PREFIX} %{IPTABLES_ETHERNET} %{IPTABLES_IP} %{IPTABLES_IP_PAYLOAD}
```

Figura 11: pipeline/patterns/iptables.pattern

Questo è il file dove viene definito il pattern **grok** utilizzato per riconoscere log di iptables. Ci permette di trasformare questo testo:

”Jan 9 13:56:35 filebeat OUTPUT: IN= OUT=eth0 MAC= SRC=172.19.0.5 DST=172.19.0.4
LEN=52 TOS=00 PREC=0x00 TTL=64 ID=64687 DF PROTO=TCP SPT=46718
DPT=5044 SEQ=455164549 ACK=2026584962 WINDOW=501 ACK URGP=0 UID=0
GID=0 MARK=0”

in questo json:


```
{
  "out_device": "eth0",
  "length": "52",
  "fragment_flags": "DF",
  "logsource": "filebeat",
  "ttl": "64",
  "mac": "",
  "dst_ip": "172.19.0.4",
  "src_ip": "172.19.0.5",
  "src_port": "46718",
  "prec": "00",
  "proto": "TCP",
  "dst_port": "5044",
  "tos": "00",
  "id": "64687",
  "in_device": "",
  "timestamp": "Jan  9 13:56:35",
  "direction": "OUTPUT"
}
```

Figura 12: grok output

2.5 Kibana

I file relativi a kibana si trovano in `/docker/config/kibana`

2.5.1 kibana.yml

```
server.name: kibana
server.host: 0.0.0.0
server.publicBaseUrl: "${KIBANA_URL}"

# Elasticsearch settings
elasticsearch.hosts: ["https://elasticsearch:9200"]
elasticsearch.ssl.certificateAuthorities: ["${CONFIG_DIR}/ca.crt"]
elasticsearch.username: "${ELASTIC_USERNAME}"
elasticsearch.password: "${ELASTIC_PASSWORD}"

# Elasticsearch monitoring settings
monitoring.ui.container.elasticsearch.enabled: true

# X-Pack Security
xpack.security.enabled: true
xpack.encryptedSavedObjects.encryptionKey: "${ENCRYPTION_KEY}"
xpack.security.encryptionKey: "${ENCRYPTION_KEY}"
xpack.reporting.encryptionKey: "${ENCRYPTION_KEY}"

# SSL settings
server.ssl.enabled: true
server.ssl.certificate: "${CONFIG_DIR}/kibana.crt"
server.ssl.key: "${CONFIG_DIR}/kibana.key"
server.ssl.certificateAuthorities: ["${CONFIG_DIR}/ca.crt"]
```

Figura 13: kibana.yml

Come per gli altri servizi vengono definite le impostazioni generali di Kibana, definendo anche le credenziali elastic e l'URL su cui verrà esposta l'applicazione web

2.6 Filebeat

I file relativi a filebeat si trovano in `/docker/config/filebeat`

2.6.1 filebeat.yml

```
name: filebeat
setup.template:
  enabled: true
output.elasticsearch.username: ${ELASTIC_USERNAME}
output.elasticsearch.password: ${ELASTIC_PASSWORD}
path.config: ${CONFIG_DIR}

filebeat.inputs:
- type: filestream
  paths:
    - "/var/log/syslog.log"

output.logstash:
  enabled: true
  hosts: ["logstash:5044"]
  ssl.certificate_authorities: [ "${CONFIG_DIR}/ca.crt" ]
  ssl.certificate: "${CONFIG_DIR}/filebeat.crt"
  ssl.key: "${CONFIG_DIR}/filebeat.key"

output.elasticsearch:
  enabled: false
  hosts: ["https://elasticsearch:9200"]
  ssl.certificate_authorities: [ "${CONFIG_DIR}/ca.crt" ]
  ssl.certificate: "${CONFIG_DIR}/filebeat.crt"
  ssl.key: "${CONFIG_DIR}/filebeat.key"
```

Figura 14: filebeat.yml

Le impostazioni generali per il servizio Filebeat.

Viene definito come input il file che vogliamo spedire a logstash: `/var/log/syslog.log` che è dove verranno salvati i log del firewall

L'output verso logstash è abilitato, mentre quello verso elasticsearch no. È possibile fare il contrario se non abbiamo bisogno di logstash per la manipolazione dei dati, ma solamente uno dei due output può essere attivo, altrimenti filebeat non parte.

2.6.2 iptable-rules.sh

Qui vengono definite le regole da applicare al firewall. Il file è un eseguibile che verrà lanciato all'avvio del container così da applicare subito le regole.

Importante è abilitare il log dei pacchetti che siamo interessati a visualizzare su kibana utilizzando il flag `-j NFLOG`.

Questo perchè docker non permette i log del kernel dei container per evitare attacchi DOS. Dobbiamo quindi simulare il log di sistema (che é dove vengono salvati normalmente i log di iptables) tramite ulogd

```
[global]
logfile="/var/log/ulogd.log"
stack=log1:NFLOG,base1:BASE,ifil:IFINDEX,ip2str1:IP2STR,print1:PRINTPKT,emu1:LOGEMU
[log1]
group=1
[emu1]
file="/var/log/syslog.log"
sync=1
```

Figura 15: ulogd.conf

Questo è il file di configurazione di ulogd.

```
# Log all packets
iptables -A FORWARD -j NFLOG --nflog-prefix "FORWARD: " --nflog-group 1
iptables -A INPUT -j NFLOG --nflog-prefix "INPUT: " --nflog-group 1
iptables -A OUTPUT -j NFLOG --nflog-prefix "OUTPUT: " --nflog-group 1
```

Figura 16: iptables.png

Con queste regole vengono loggati tutti i pacchetti che passano per la macchina.

3 Monitoraggio

3.1 Avvio

Per poter utilizzare i servizi va prima popolato il file **docekr/.env**.

Quindi dopo aver installato ed avviato Docker, con **docker/cyber.sh setup** vengono generati i certificati per la crittografia.

Una volta che termina il comando precedente possiamo avviare i container con **docker/cyber.sh up** e possiamo controllarne lo stato con **docker ps**, **docker logs** o con software di monitoraggio come **lazydocker**, **docker desktop** o **portainer**.

3.2 Kibana

Una volta che i container sono pronti possiamo visualizzare la pagina web di kibana su **https://localhost:5601**

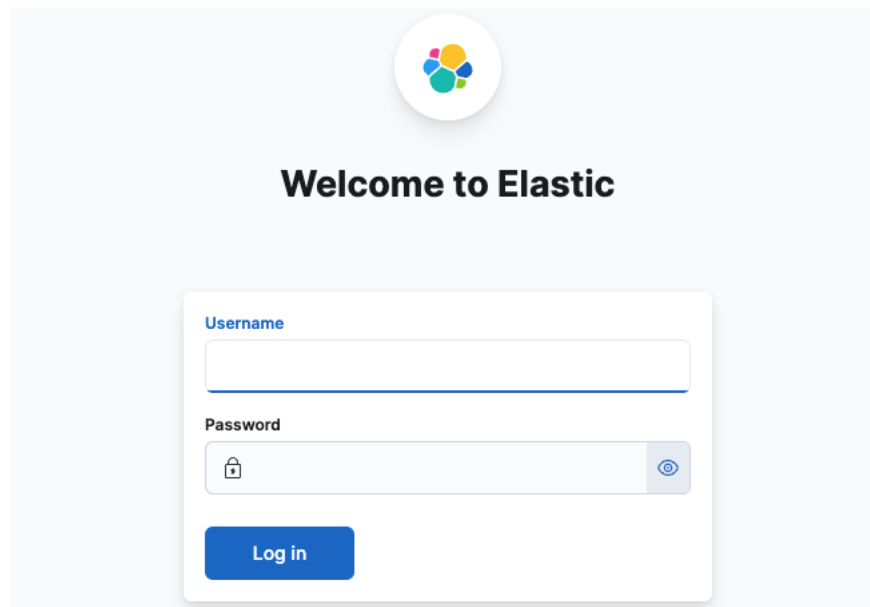


Figura 17: Schermata di login

Utilizziamo le credenziali definite nel file **.env** per effettuare il login e possiamo vedere la homepage di Kibana.

Per fare qualche prova possiamo popolare il file di log di iptables in vari modi, ad esempio:

```
curl localhost:80
```

Aperto quindi il menù in alto a sinistra, clicchiamo su **Discover** e infine su **Index Management**, vediamo che i dati di filebeat stanno arrivando. Qui è possibile vedere i dati dei vari indici che arrivano a elasticsearch, nel nostro caso l'unico indice è quello generato da logstash.

Ora possiamo procedere a creare un **Index Pattern**. Per farlo andiamo su **Index Patterns**, inseriamo un nome che corrisponda al nostro indice e selezioniamo l'unico timestamp disponibile: questo servirà a indicizzare in serie temporale i vari log.

Cliccando su **Create Index Pattern** e tornando su **Discover** possiamo visualizzare i dati in arrivo.

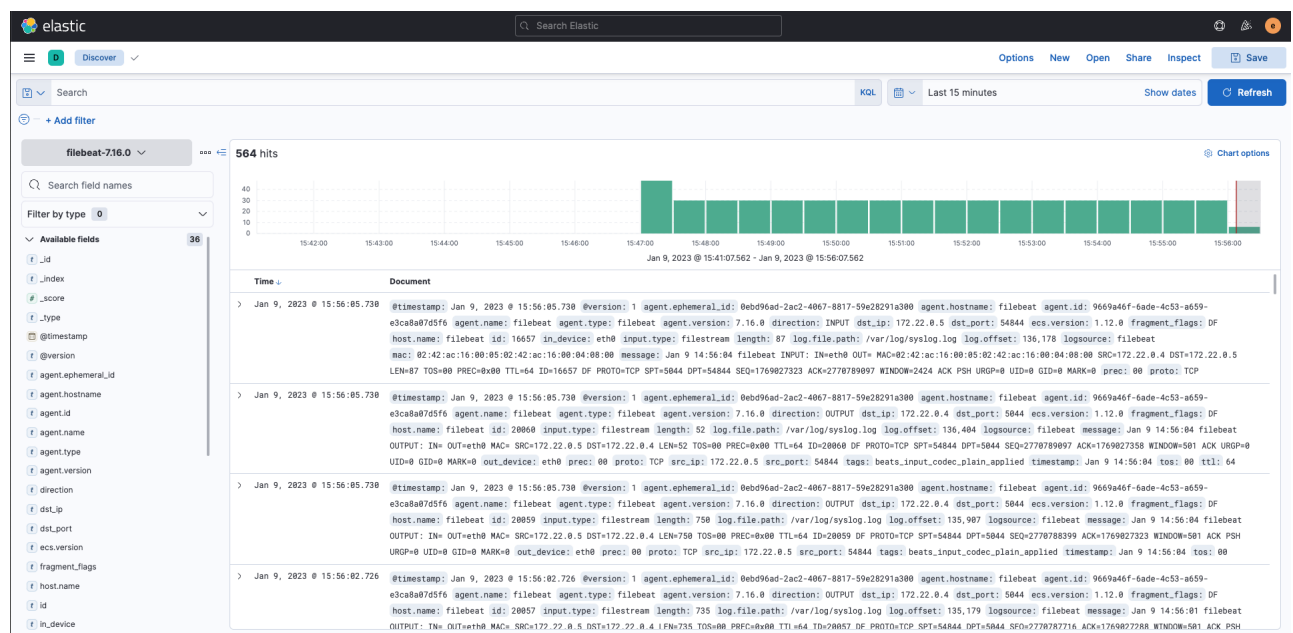


Figura 18: Log di iptables visualizzati su Kibana

Qui possiamo utilizzare **KQL (Kibana Query Language)** per visualizzare i dati che vogliamo, con la possibilità di utilizzare i campi definiti nel filtro grok di logstash per i filtri.

Per esempio possiamo vedere solamente i pacchetti passati per la porta 80 con questa query:

```
dst_port : 80 or src_port : 80
```

E vediamo solamente i due pacchetti, uno in entrata ed uno in uscita.

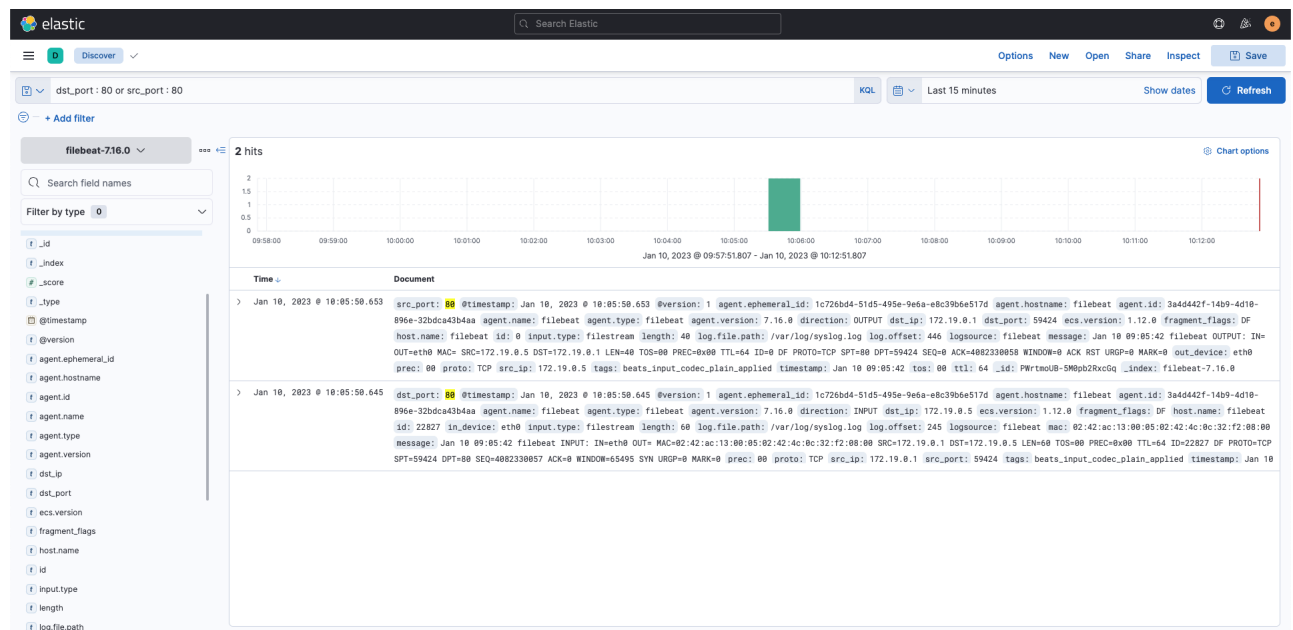


Figura 19: Risultato della Query

4 Analisi dei Dati

4.1 Port Scan Attack

Il **Port Scan** è una delle principali tecniche di **ricognizione** che gli attaccanti usano per raccogliere dati sulla macchina vittima.

Consiste nello scansionare numerose porte sull'host da attaccare per ricevere informazioni su quali sono aperte e quali chiuse. Da queste informazioni spesso si riesce a riconoscere quali servizi sono attivi sulla macchina per poi decidere come continuare l'attacco.

Il Port Scan può essere utilizzato in maniera ancora più aggressiva per fare **Banner Grabbing**. Leggendo le risposte che i servizi inviano dopo una connessione si può ricorrere allo specifico software utilizzato e spesso anche alla versione. Questo combinato con script di attacco trovati in rete o semplicemente su Metasploit può portare a gravi attacchi anche da parte di persone poco esperte.

4.2 L'attacco

Proviamo ad eseguire un port scan attack sulla nostra macchina contenete il firewall. Dato che la macchina si trova su un container dove poche porte sono esposte, lo facciamo dall'interno. In una situazione reale il firewall avrà accesso a tutta la rete della macchina da proteggere e riusciremo a riconoscere attacchi anche dall'esterno.

Utilizziamo `docker exec` per lanciare il comando all'interno del container

```
docker exec -it filebeat sh -c "apt install nmap -y"
```

nmap serve proprio a fare port scanning.

```
docker exec -it filebeat sh -c "nmap localhost -Pn -p-"
```

Il flag **-Pn** evita che non venga neanche tentata la connessione su una porta in caso di risposta negativa ad un ping. Nmap senza questo flag prova prima un ping sulla porta, se non c'è risposta non tenta la connessione. Spesso alcuni servizi non rispondono a ping proprio per evitare attacchi di port scan

-p- serve per effettuare lo scan su tutte le porte

Guardando su Kibana possiamo notare che sono stati scritte più di 127000 righe di log durante lo scan.

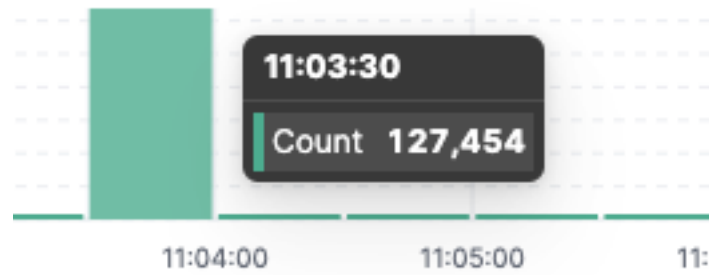


Figura 20: Log durante lo scan

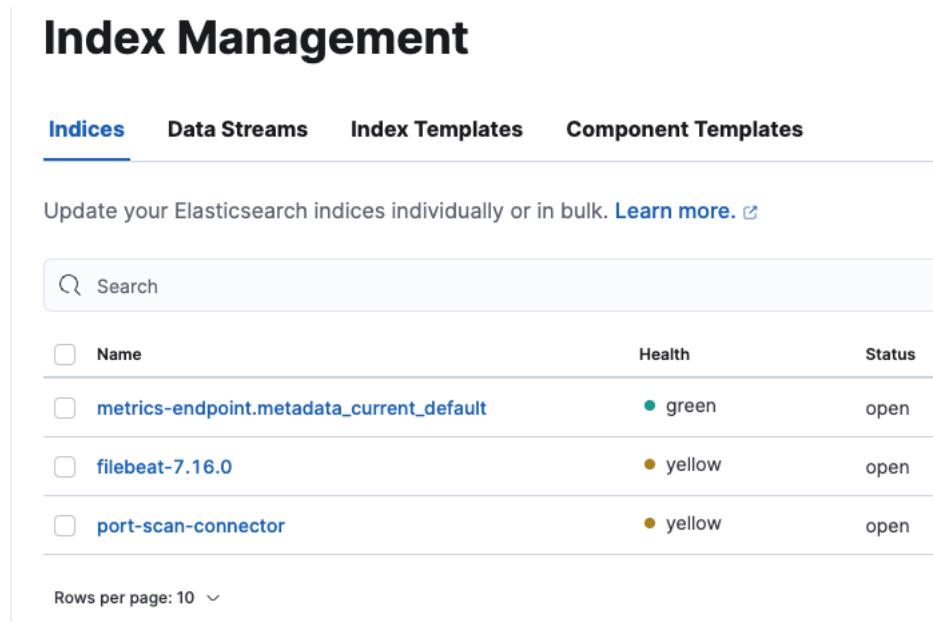
4.3 Regole

Su Kibana possiamo aggiungere delle regole di controllo sui nostri dati, queste regole poi possono generare degli alert e notificare in varie modalità un utente dei proprio risultati.

Per prima cosa dobbiamo creare un **connector**, ovvero un metodo per ricevere le segnalazioni. Tra i vari connector possibili c'è la possibilità di scegliere email, slack, microsoft teams e molti altri. Noi scegliamo **index**, ovvero vogliamo creare un indice nel nostro database elasticsearch dove verranno salvati i risultati della regola.

Per creare il connector basta andare nel menù in alto a sinistra e cliccare su **Stack Management**, cercare poi la voce **Rules and Connectors**. Qui possiamo andare sulla pagina dei connectors e fare **Create Connector**. Ora gli diamo un nome, configuriamo quello che deve scrivere e salviamo.

Dobbiamo infine creare l'**Index Pattern** come abbiamo fatto per filebeat. Dovremmo ora vedere il nostro index della pagina **Discovery**, ovviamente vuoto.



The screenshot shows the 'Index Management' page in Kibana. It has a header with four tabs: 'Indices' (selected), 'Data Streams', 'Index Templates', and 'Component Templates'. Below the tabs is a message: 'Update your Elasticsearch indices individually or in bulk. [Learn more.](#)'. There is a search bar with a magnifying glass icon and the text 'Search'. Below the search bar is a table with three columns: 'Name', 'Health', and 'Status'. The table contains three rows of indices. Each row has a checkbox in the first column. The first row is 'metrics-endpoint.metadata_current_default' with a green dot for health and 'open' status. The second row is 'filebeat-7.16.0' with a yellow dot for health and 'open' status. The third row is 'port-scan-connector' with a yellow dot for health and 'open' status. At the bottom left, there is a label 'Rows per page: 10' with a dropdown arrow.

<input type="checkbox"/>	Name	Health	Status
<input type="checkbox"/>	metrics-endpoint.metadata_current_default	● green	open
<input type="checkbox"/>	filebeat-7.16.0	● yellow	open
<input type="checkbox"/>	port-scan-connector	● yellow	open

Rows per page: 10 ▾

Figura 21: Index Mapping con port-scan-connector

Per creare la regola torniamo nel menù in alto a sinistra e sotto **security** clicchiamo su **Overview**, ora sempre a sinistra troviamo **Rules** per andare alla pagina delle regole presenti. Facciamo **Create Rule**

Volgiamo creare una regola per identificare Port Scan Attack. Quindi vogliamo essere notificati per un possibile attacco quando un determinato ip fa **numeroso** richieste provenienti dalla stessa porta verso porte diverse.

Rule type

Threshold
Aggregate query results to detect when number of matches exceeds threshold.
✓ Selected

Index patterns [Reset to default index patterns](#)
filebeat-* ×
Enter the pattern of Elasticsearch indices where you would like this rule to run. By default, these will include index patterns defined in Security Solution advanced settings.

Custom query [Import query from saved timeline](#)
@timestamp < now-1m KQL

+ Add filter

Group by
src_ip.keyword × src_port.keyword × × ▾ >= 1
Select fields to group by. Fields are joined together with 'AND'

Count
dst_port.keyword × ▾ >= 500
Select a field to check cardinality

Unique values

Timeline template
None ▾
Select which timeline to use when investigating generated alerts.

Figura 22: Creazione della regola

Questa regola ci notificherà quando un indirizzo ip fa richieste su almeno 500 porte diverse in meno di un minuto. Tra le varie impostazioni possiamo:

- Scegliere un **Severity Level** con un punteggio di rischio.
- Scegliere ogni quanto vogliamo che la regola venga lanciata. Impostiamo **1 minuto**, così da rimanere sempre coperti.

- Aggiungere azioni da eseguire. Nel nostro caso colleghiamo il connector creato in precedenza e selezioniamo "ad ogni esecuzione della regola", così facendo verrà creato un log solamente quando la regola risulta positiva. Dobbiamo anche specificare cosa scrivere sul file dell'index, scegliamo quello consigliato dalla documentazione, ma aggiungiamo il campo "**alerts**" per poter visualizzare i dati dell'attacco.

Actions

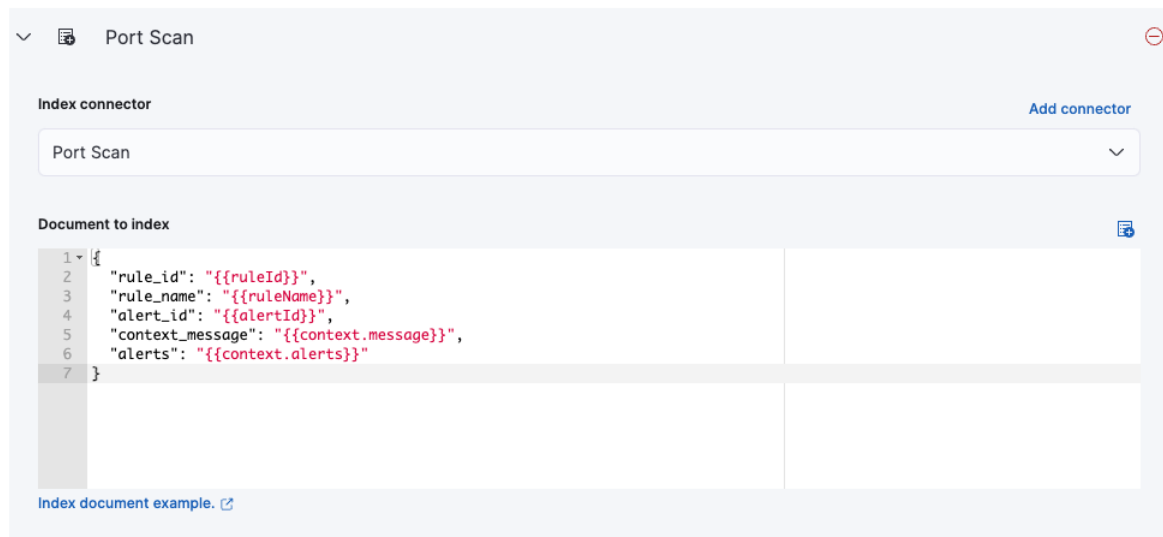


Figura 23: Azioni della regola

4.4 Test

Dopo aver abilitato la regola proviamo se funziona. Come prima facciamo un port scan con nmap.

```
docker exec -it filebeat sh -c "nmap localhost -Pn -p-"
```

Se controlliamo la sezione **Discover** dopo circa un minuto possiamo vedere un nuovo log che ci indica un possibile **tentato attacco Port Scan**

Expanded document

View single document

Table JSON

Actions	Field	Value
	_id	3hVboIU81xp80Z49GduK
	_index	port-scan-connector
	_score	1
	_type	_doc
	alert_id	3c8b22e0-9199-11ed-84a3-f7b55f531183
	alerts	>
	context_message	(empty)
	rule_id	(empty)
	rule_name	(empty)

Figura 24: Log di alert

Dentro il campo "alerts" possiamo vedere che l'attacco proviene da "127.0.0.1"!

4.5 Altre Regole

Con i dati collezionati in questo modo si possono poi creare altri tipi di alert esterni a Kibana, oppure creare dashboard dove è possibile visualizzare più semplicemente quello che sta accadendo sulla rete.

Possiamo anche andare a creare regole diverse per generare alert basati su altri tipi di query, come:

- **Custom Query:** Per usare **Kibana Query Language** oppure il linguaggio **Lucene** per creare query **personalizzate** sui file di log.
- **Machine Learning:** Utilizza machine learning per fare **anomaly detection** sui dati, ovvero c'è un alert quando i dati **deviano** da quelli che sono considerati normali.
- **Threshold:** Quello che è stato utilizzato per controllare il Port Scan attack. Permette di contare il numero di record che corrisponde ad una certa query e genera un alert quando il loro numero supera una certa soglia.
- **Event Correlation:** Utilizza **EQL (Event Query Language)**, un linguaggio di query per dati generati da eventi in time-series, per generare alert.
- **Indicator Match:** Permette di mettere in correlazione più indici per creare regole.

5 Fonti

- Documentazione Elastic:

<https://www.elastic.co/guide/index.html>

- elk-tls-docker:

<https://github.com/swimlane/elk-tls-docker>

- Running and Debugging Iptables inside Docker container:

<https://hangarau.space/running-and-debugging-iptables-inside-a-docker-container>

- Iptables Grok pattern:

<https://gist.github.com/Caligatio/1c1ef69dd720186cfacf95c1cd8ea85d>

- Kibana Aggregations Explained:

<https://www.youtube.com/watch?v=j-eCKDhj-0s>