Mise en place d’une supervision de sécurité SSH avec Fail2Ban, Prometheus et Grafana

1. Introduction et contexte

Dans un contexte de cybersécurité moderne, la supervision des événements de sécurité est un élément fondamental d’un système d’information. La détection des tentatives d’attaque par force brute sur des services exposés, notamment SSH, constitue un cas d’usage classique et représentatif d’un mini-SOC (Security Operations Center).

Ce projet a consisté à concevoir et implémenter une architecture de supervision permettant :

- de détecter les tentatives de connexion SSH malveillantes,
- de mesurer les bannissements effectués par Fail2Ban,
- d’exposer ces métriques au format Prometheus,
- de les visualiser dynamiquement dans Grafana.

Le projet a été réalisé dans un environnement Linux sous WSL2, avec orchestration Docker.

2. Objectifs pédagogiques et techniques

2.1 Objectifs pédagogiques

- Comprendre le fonctionnement de Fail2Ban.
- Mettre en œuvre un exporter Prometheus personnalisé.
- Configurer Prometheus pour le scraping de métriques.
- Concevoir un dashboard Grafana de supervision sécurité.
- Diagnostiquer et résoudre des incidents liés à Docker et aux volumes persistants.

2.2 Objectifs techniques

- Superviser les métriques suivantes :
  - `fail2ban_failed_total`
  - `fail2ban_banned_total`
- Mettre en place un cycle complet :
  - Log → Détection → Exposition métrique → Scraping → Visualisation.

3. Architecture du système

3.1 Composants

Le système repose sur quatre briques principales :

1. Fail2Ban

- Analyse les logs SSH (`/var/log/auth.log`)
- Détecte les tentatives échouées
- Bannit une IP après `maxretry` tentatives dans `findtime`
- Durée du bannissement : `bantime`

Configuration observée :

- `maxretry = 3`
- `findtime = 600s`
- `bantime = 600s`

```
narm@pcn:~/projets/mini-projets/prometheus-monitoring$ sudo fail2ban-client get sshd
findtime
[sudo] password for narm:
600
narm@pcn:~/projets/mini-projets/prometheus-monitoring$ sudo fail2ban-client get sshd
bantime
600
```

```
narm@pcn:~/projets/mini-projets/prometheus-monitoring$ sudo fail2ban-client get sshd
maxretry
3
```

2. Exporter Python personnalisé

Un script Python expose les métriques au format Prometheus via HTTP sur le port **9188**.

Il permet à Prometheus de collecter :

fail2ban_failed_total
fail2ban_banned_total

Cette étape est cruciale car Fail2Ban n'expose pas nativement ses métriques au format Prometheus.

3. Prometheus

Prometheus scrappe les métriques toutes les 15 secondes :

scrape_interval: 15s

Configuration du target :

targets: ['172.20.246.247:9188']

L'utilisation de l'IP réelle a été nécessaire après un incident réseau lié à `host.docker.internal`.

4. Grafana

Grafana permet la visualisation des métriques via :

- Data source : `http://prometheus:9090`
- Panels de type **Stat**

Dashboard final :

- Nombre total de bannissements
- Nombre total de tentatives échouées

4. Structure du projet et fichiers de configuration

4.1 Arborescence du projet

L'organisation du projet repose sur une séparation claire des composants Prometheus, Grafana et de l'exporter personnalisé.

Arborescence :

```
├── docker-compose.yml
├── grafana
├── prometheus
│   └── prometheus.yml
└── prometheus-fail2ban-exporter
    ├── README.md
    ├── fail2ban-exporter.py
    └── requirements.txt
```

Description des dossiers :

- **docker-compose.yml** : orchestration des services Docker (Prometheus et Grafana)
- **prometheus/** : configuration de Prometheus
- **grafana/** : données persistantes via volume Docker
- **prometheus-fail2ban-exporter/** : exporter Python personnalisé

4.2 Fichier docker-compose.yml

Ce fichier permet de lancer les services Prometheus et Grafana dans des conteneurs Docker.

Extrait :

```
version: '3'

services:
  prometheus:
    image: prom/prometheus
    container_name: prometheus
    volumes:
      - ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml
      -
/var/lib/prometheus/node-exporter:/var/lib/prometheus/node-exporter
    ports:
      - "9090:9090"

  grafana:
    image: grafana/grafana
    ports:
      - "3000:3000"
    volumes:
      - grafana-storage:/var/lib/grafana

volumes:
  grafana-storage:
```

Rôle :

- Démarre Prometheus sur le port 9090
- Démarre Grafana sur le port 3000
- Monte un volume persistant pour Grafana

4.3 Fichier prometheus.yml

Ce fichier configure les cibles que Prometheus doit surveiller.

```
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'fail2ban'
    static_configs:
      - targets: ['172.20.246.247:9188']
```

Explication :

- Prometheus collecte les métriques toutes les 15 secondes.
- La cible correspond à l'exporter Python exposant les métriques Fail2Ban.

4.4 Exporter Python (fail2ban-exporter.py)

L'exporter :

- Interroge l'état de Fail2Ban
- Convertit les informations en métriques Prometheus
- Expose ces métriques sur le port 9188

```python
#!/usr/bin/env python3
import re
import subprocess
import time
from prometheus_client import Gauge, start_http_server

PORT = 9188

parseKeys = {
    'Currently failed:': ('fail2ban_failed_current', 'Number of currently failed connections.'),
    'Total failed:': ('fail2ban_failed_total', 'Total number of failed connections.'),
    'Currently banned:': ('fail2ban_banned_current', 'Number of currently banned IP addresses.'),
    'Total banned:': ('fail2ban_banned_total', 'Total number of banned IP addresses.')
}

pattern = re.compile(r'(' + '|'.join(parseKeys.keys()) + r')\s*(\d*)')

# Création des métriques (une seule fois)
gauges = {}
for key, value in parseKeys.items():
    gauges[key] = Gauge(value[0], value[1], ['jail'])

def collect_metrics():
    process = subprocess.Popen(['fail2ban-client', 'status'], stdout=subprocess.PIPE)
    response = process.communicate()[0].decode('utf-8')
    match = re.search(r'.+Jail list:\s+(.+)$', response)

    if not match:
        return

    jails = match.group(1).split(", ")

    for jail in jails:
        process = subprocess.Popen(['fail2ban-client', 'status', jail], stdout=subprocess.PIPE)
        response = process.communicate()[0].decode('utf-8')
        matches = re.findall(pattern, response)

        for m in matches:
            gauges[m[0]].labels(jail=jail).set(float(m[1]))

if __name__ == "__main__":
    start_http_server(PORT)
    print(f"[OK] fail2ban exporter listening on http://0.0.0.0:{PORT}/metrics")

    while True:
        collect_metrics()
```

Exemple de métriques exposées :

`fail2ban_failed_total`
`fail2ban_banned_total`

Ce composant est central dans l’architecture car Fail2Ban ne fournit pas nativement d’endpoint compatible Prometheus.

5. Flux de données

Le pipeline complet est le suivant :

Tentatives SSH
↓
/var/log/auth.log
↓
Fail2Ban
↓
Exporter Python (port 9188)
↓
Prometheus
↓
Grafana

Cette architecture reproduit un mini système de supervision de sécurité.

6. Vérification et validation du fonctionnement

Afin de s'assurer du bon fonctionnement de l'architecture mise en place, plusieurs vérifications ont été réalisées à différents niveaux du pipeline.

6.1 Vérification des services Docker

Après le lancement via :

```
docker-compose up -d
```

Les conteneurs ont été vérifiés avec :

```
docker ps
```

Deux services principaux devaient être actifs :

- Prometheus (port 9090)
- Grafana (port 3000)

6.2 Vérification de l'interface Prometheus

L'accès à l'interface web de Prometheus a été validé via :

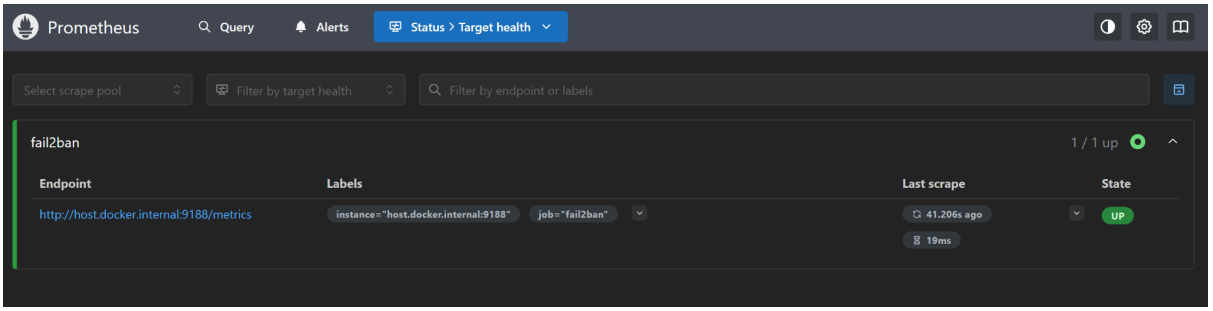
`http://localhost:9090/`

Cela permet :

- de vérifier que le serveur Prometheus est opérationnel,
- d'interroger directement les métriques exposées.

Dans l'onglet **Status > Targets**, la cible `fail2ban` devait apparaître avec l'état :

`State: UP`



Cela confirme que :

- Prometheus parvient à joindre l'exporter,
- le scraping des métriques est actif.

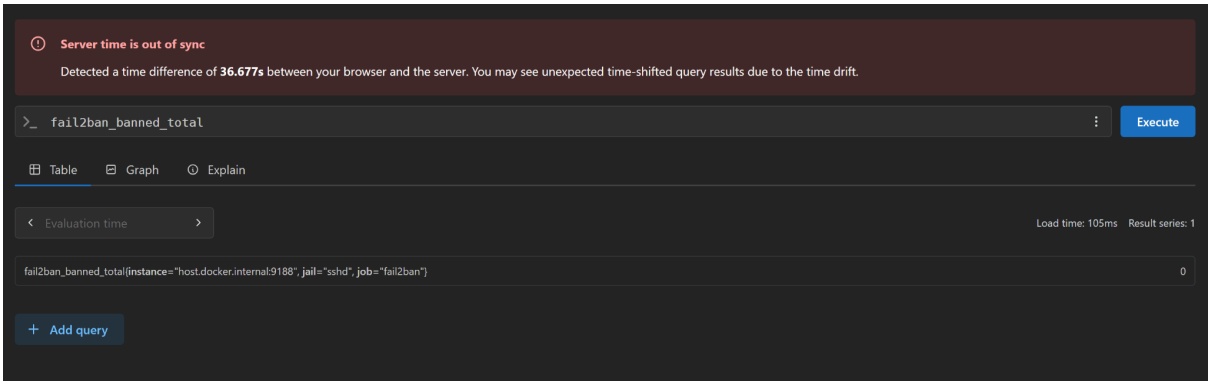
6.3 Vérification de l'existence des métriques

Dans l'onglet **Query** de Prometheus, la requête suivante a été exécutée :

`fail2ban_banned_total`

La présence d'un résultat (même égal à 0) confirme :

- que la métrique est correctement exposée,
- que Prometheus l'a bien enregistrée.



6.4 Vérification de l'interface Grafana

L'accès à Grafana a été validé via :

`http://localhost:3000/`

Après configuration de la data source Prometheus :

`http://prometheus:9090`

Les panels suivants ont été créés :

- Nombre total de bannissements
- Nombre total de tentatives échouées

La cohérence entre :

- la valeur affichée dans Prometheus,
- et la valeur affichée dans Grafana,

a permis de confirmer la validité complète du pipeline.

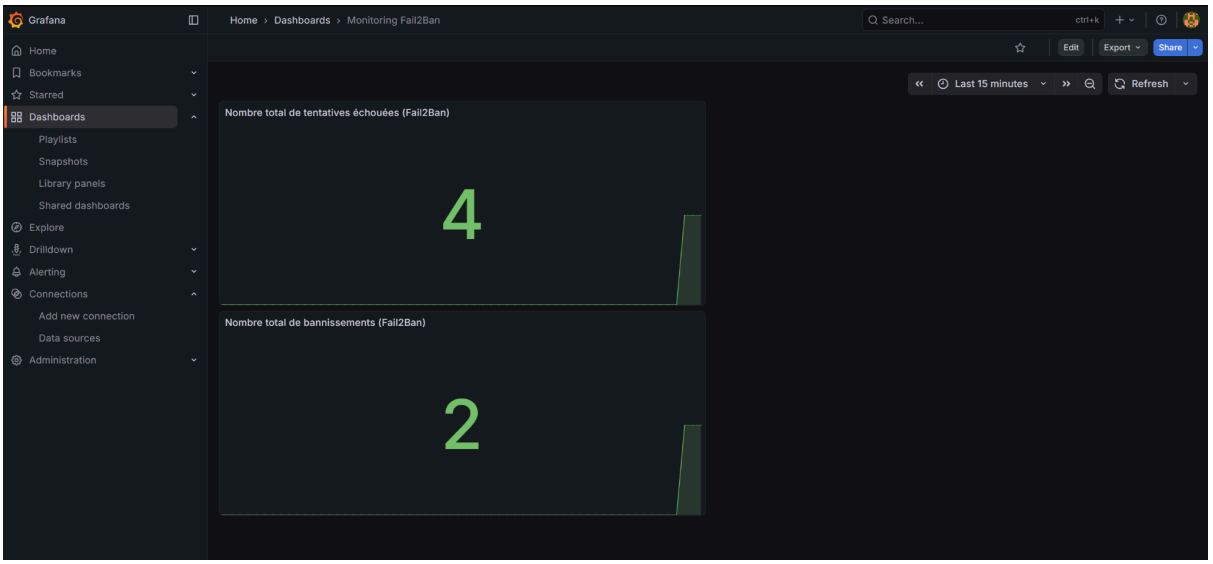
6.5 Validation finale par test de bannissement

La commande suivante a été utilisée pour simuler un bannissement :

```
sudo fail2ban-client set sshd banip 8.8.8.8
```

Résultat observé :

- Incrémentation de `fail2ban_banned_total`
- Target Prometheus toujours en état UP
- Mise à jour en temps réel du panel Grafana



Cette étape confirme le bon enchaînement :

Fail2Ban → Exporter → Prometheus → Grafana.

7. Incident majeur : redémarrage du système

Après redémarrage de l'ordinateur, plusieurs dysfonctionnements sont apparus.

7.1 Problème 1 : Target Prometheus en DOWN

Symptôme :

Status: DOWN / UNKNOWN

Analyse :

- `host.docker.internal` ne résolvait plus correctement.
- Sous WSL2, ce mapping est instable après reboot.

Solution :

- Identification de l'IP réelle via `hostname -I`
- Remplacement du target par l'IP explicite.

7.2 Problème 2 : Erreur "out-of-order samples"

Logs Prometheus :

Error on ingesting out-of-order samples

Cause :

- L'exporter redémarré produisait des métriques repartant à 0.
- Prometheus conservait en mémoire les anciennes valeurs (WAL).
- Conflit temporel détecté.

Solution :

- Suppression du volume Prometheus
- Redémarrage propre
- Reconstruction de la base TSDB

7.3 Problème 3 : Perte des dashboards Grafana

Observation :

No data sources found

Analyse :

Le volume Docker `grafana-storage` avait été recréé :

CreatedAt: 2026-02-19T12:49:57

Conséquence :

- Perte des dashboards
- Perte des data sources
- Perte de grafana.db initiale

Solution :

- Reconfiguration complète de la data source
- Recréation du dashboard

8. Difficultés spécifiques à WSL2

WSL2 a introduit plusieurs contraintes :

- Backend systemd instable
- Interaction partielle avec iptables
- Difficulté à bannir localhost (127.0.0.1)
- Comportement atypique de Fail2Ban en environnement virtualisé

Cela a nécessité :

- Tests via `fail2ban-client set sshd banip`
- Tests via IP réseau réelle
- Diagnostic approfondi des logs

9. Validation du fonctionnement

La validation du pipeline complet a été réalisée en plusieurs étapes.

9.1 Simulation manuelle d'un bannissement

Dans un premier temps, un bannissement a été simulé manuellement via la commande :

- `sudo fail2ban-client set sshd banip 8.8.8.8`

Cette commande a permis de valider :

- L'incrémentation de la métrique `fail2ban_banned_total`
- Le scraping correct par Prometheus

- La mise à jour en temps réel du panel Grafana

Cette étape a confirmé la cohérence complète du pipeline :

Fail2Ban → Exporter → Prometheus → Grafana

Cependant, cette méthode restait une simulation contrôlée et non une attaque réelle.

9.2 Simulation réelle d'attaque par force brute

Afin de tester le système dans des conditions plus réalistes, une attaque simulée par tentatives SSH répétées a été réalisée à l'aide de la commande suivante :

- `for i in {1..5}; do sshpass -p "wrongpass" ssh -o StrictHostKeyChecking=no fakeuser@172.20.246.247; done`

Cette commande :

- Automatise l'envoi de mots de passe incorrects
- Génère des échecs d'authentification SSH réels
- Déclenche le mécanisme de détection de Fail2Ban

Résultat observé :

```
Permission denied, please try again.

...

ssh: connect to host 172.20.246.247 port 22: Connection refused
```

Puis :

```
sudo fail2ban-client status sshd
```

Retour :

```
Currently failed: 1
Total failed: 4
Currently banned: 1
Total banned: 2
Banned IP list: 172.20.246.247
```

Cela démontre que :

- Les tentatives SSH ont été correctement enregistrées dans `/var/log/auth.log`
- Fail2Ban a détecté le dépassement de `maxretry`
- L'IP 172.20.246.247 a été bannie automatiquement
- La métrique `fail2ban_banned_total` a été incrémentée
- Prometheus a scrappé la nouvelle valeur
- Grafana a mis à jour le panel en conséquence

9.3 Validation finale

La réussite du bannissement automatique via SSH confirme que le système fonctionne dans un scénario réaliste d'attaque par force brute.

Contrairement à la simulation manuelle, cette méthode valide :

- Le fonctionnement complet de la chaîne de détection
- L'interaction réelle entre SSH, Fail2Ban et l'exporter
- La supervision dynamique via Prometheus et Grafana

Ainsi, le pipeline de supervision sécurité est pleinement opérationnel.

10. Analyse critique

Points positifs

- Architecture modulaire et évolutive
- Exporter personnalisé fonctionnel
- Diagnostic réseau Docker maîtrisé
- Gestion avancée des volumes persistants
- Compréhension du fonctionnement interne de Prometheus (TSDB, WAL)
- Validation du pipeline via simulation d'attaque réelle

Limites

- Environnement WSL2 non idéal pour tester iptables
- Absence de tests depuis une machine externe réelle
- Pas d'alerting configuré
- Dépendance à une IP spécifique dans la configuration

11. Perspectives d’amélioration

- Ajout d’Alertmanager
- Intégration Loki pour logs centralisés
- Déploiement sur VM Linux native
- Mise en place d’un second service surveillé
- Ajout d’un système d’alertes mail ou Slack

12. Conclusion

Ce projet a permis de mettre en œuvre une architecture complète de supervision sécurité moderne basée sur des outils largement utilisés en production.

Au-delà de l’implémentation fonctionnelle, la résolution des incidents rencontrés (réseau Docker, volumes, WAL, redémarrage système) a permis d’approfondir la compréhension des mécanismes internes de :

- Docker
- Prometheus
- Fail2Ban
- Grafana

Le système final constitue une base solide pour évoluer vers une architecture SOC plus complète.