Projet Master : SIEM Minimaliste

Master 2 Cyber

Livrable n°1

Sablayrolles Adrien

FACULTÉ
GESTION, ÉCONOMIE,
SCIENCES
Université Catholique
de Lille 1875

1. Description

Le projet SIEM Minimaliste a pour objectif de développer un Security Information and Event Management en Python, destiné à collecter, analyser et corréler des événements de sécurité provenant de différentes sources (journaux système, réseau, etc.).

Les points principaux du projet :

- **La collecte** de logs (Syslog, Windows Event Logs, journaux SSH, etc.).
- **L'analyse** des données (détection d'anomalies, règles statiques).
- **La visualisation** et le suivi des alertes via un tableau de bord web.
- **La notification** et la gestion des alertes.
- **La portabilité** grâce à Docker. (facultatif)

2. Planning

Phase	Objectif principal	Durée estimée
Phase 1 : Conception	- Définir l'architecture globale (collecte, analyse, etc.) - Choisir les technologies (Python, Docker)	1 semaine
Phase 2 : Collecte	- Mettre en place les modules de collecte (Evtx, Syslog, Windows, SSH) - Normaliser les formats de logs	2 semaines
Phase 3 : Analyse	- Implémenter des règles statiques basiques - Mettre en place un module de détection d'anomalies	2 semaines
Phase 4 : Alerting	- Configurer l'envoi d'alertes - Gérer la gravité et le statut des alertes	1 semaine
Phase 5 : Dashboard	- Développer une interface web (Python,Html, CSS, JS) - Visualiser les logs et alertes avec le minimum de délai	2 semaines
Phase 6 : Tests & Déploiement	- Réaliser les tests unitaires et d'intégration - Conteneurisé l'application via Docker - Documentation et mise en production	1 semaine

3. Cahier des charges

3.1 Objectifs fonctionnels

1. **Collecte de logs**
    - Doit supporter plusieurs sources (ex. Evtx, Syslog, Windows Event Logs).
2. **Analyse des événements**
    - **Règles statiques** : détections basées sur des seuils ou signatures (nombre de connexions SSH ratées, etc.).
    - **Détection d'anomalies** : algorithmes basiques pour identifier des comportements inhabituels.
3. **Gestion des alertes**
    - Génération d'alertes (niveaux de criticité : bas, moyen, élevé).
    - Notification (pour les niveaux d'alerte élevé)
    - Interface de gestion pour filtrer et classer les alertes.
4. **Tableau de bord**
    - Interface web (Python, HTML, JS) permettant de visualiser :
        - Les alertes en temps réel.
    - Possibilité de filtrer par source de log, type d'événement, etc .
5. **Rapports**
    - Génération de rapports récapitulatifs.
6. **(facultatif) Sécurité & Conformité**
    - Authentification et contrôle d'accès à l'interface

- Encryption des données sensibles

3.2 Contraintes techniques

1. **Langage & Framework**
    - Développement en Python (3.9+ de préférence).
    - Partie web Python,Html, CSS, JS.
2. **Base de données**
    - SQLite pour un usage léger ou PostgreSQL pour un déploiement plus robuste.
    - Organisation structurée pour stocker logs et alertes.
3. **Performances & Scalabilité**
    - Gestion d'un volume modéré de logs dans un premier temps (quelques milliers d'événements par jour).
4. **Déploiement & Docker**
    - Fournir un Dockerfile + docker-compose pour lancer l'application, la base de données et les services requis.
    - Scripts de déploiement et de mise à jour.
5. **Tests et Qualité**
    - Tests unitaires (collecte, analyse, alerting).
    - Tests d'intégration couvrant le cycle complet (ingestion → détection → alerte).
    - Rapport de couverture de tests.

3.3 Livrables

1. **Code source** (Github) : arborescence claire (collecte, analyse, alerting, etc.).
2. **Documentation** :
    - Manuel d'installation et de configuration.
    - Explication rapide sur l'implémentation des règles et du modèle ML.
3. **Images Docker** :
    - Conteneur principal (application Python).
    - Base de données et éventuellement un conteneur pour un outil de visualisation ou un agent de collecte.