Institut Supérieur des Études Technologiques (ISET Nabeul)

Rapport de Projet : Système Unifié de Détection de Menaces en Temps Réel

Conception et Implémentation d'une Plateforme de Détection Hybride basée sur Apache Spark Structured Streaming

Auteurs : Ballegi Ala, Aounallah Oussama

Classe : RSI 31

Année Universitaire : 2025/2026

https://github.com/BALLEGI/Real-Time-Threat-Analytics

Ballegi Ala
Aounallah Oussama
RSI31

Table des Matières

Ballegi Ala
Aounallah Oussama
RSI31

1. Sommaire Exécutif

Face à l'augmentation exponentielle du volume et de la sophistication des cybermenaces, les systèmes de détection d'intrusion traditionnels, souvent basés sur des signatures statiques, montrent leurs limites. Ils peinent à identifier les attaques "zero-day", les techniques d'obfuscation et les campagnes de fraude à grande échelle en temps réel. Ce rapport présente la conception, l'implémentation et l'évaluation d'un Système Unifié de Détection de Menaces visant à pallier ces faiblesses.

Notre solution s'appuie sur une architecture de traitement de flux (streaming) moderne et évolutive, construite autour d'Apache Spark Structured Streaming. Elle intègre une approche de détection hybride qui combine trois piliers complémentaires :

- La détection par signatures, pour identifier rapidement des menaces connues (injections SQL, XSS, force brute, etc.) via des expressions régulières optimisées.
- L'intégration de Threat Intelligence (TI), pour bloquer proactivement les entités (adresses IP, User-Agents) déjà répertoriées comme malveillantes, grâce à un mécanisme de diffusion efficace (broadcast) au sein du cluster Spark.
- La détection heuristique basée sur l'IA, pour découvrir des anomalies comportementales telles que l'obfuscation de code (via le calcul de l'entropie), les tentatives d'injection (ratio de caractères spéciaux) et l'exfiltration de données.

Le pipeline ingère des flux de données hétérogènes (logs système syslogs et alertes de fraude fraud_alerts) depuis Apache Kafka, les enrichit et les analyse en continu. Les alertes pertinentes sont indexées dans Elasticsearch, permettant une visualisation, une investigation et une corrélation rapides via des tableaux de bord Kibana.

Les résultats démontrent la capacité du système à détecter une large gamme d'attaques en temps réel avec une faible latence. Ce projet constitue une base robuste et flexible pour un centre d'opérations de sécurité (SOC) moderne, offrant une visibilité accrue et une réponse aux incidents plus rapide.

2. *Introduction*

2.1. Contexte et Problématique

Le paysage de la cybersécurité a radicalement changé. Les adversaires ne se contentent plus d'attaques simples et isolées ; ils déploient des campagnes complexes, multi-vectorielles et souvent automatisées. Le volume de données générées par les infrastructures IT (serveurs, applications, équipements réseau) est devenu massif, rendant l'analyse manuelle impossible.

Dans ce contexte, les systèmes de gestion des informations et des événements de sécurité (SIEM) traditionnels font face à plusieurs défis majeurs :

- Latence de détection : L'approche par batch, où les données sont analysées à intervalles réguliers, introduit un délai critique entre l'occurrence d'une attaque et sa détection, laissant une fenêtre d'opportunité large aux assaillants.
- Dépendance aux signatures : La détection basée uniquement sur des signatures est inefficace contre les attaques "zero-day", les variants de malwares et les techniques d'obfuscation (payloads Base64, shellcode encodé, etc.).
- Manque de contexte : Les alertes sont souvent générées de manière isolée, sans corrélation avec des renseignements externes (Threat Intelligence) ou des anomalies comportementales, ce qui conduit à un grand nombre de faux positifs et à la fatigue des analystes.
- Problèmes d'évolutivité : L'architecture de nombreux SIEM ne parvient pas à suivre la croissance du volume de logs, entraînant des pertes de données ou des dégradations de performance.

La problématique centrale est donc de construire une plateforme capable de traiter des flux de données en temps réel, de combiner intelligemment plusieurs méthodes de détection et de s'adapter à l'évolution des menaces, le tout de manière évolutive et performante.

2.2. Objectifs du Projet

Pour répondre à cette problématique, ce projet vise à atteindre les objectifs suivants :

- Concevoir une architecture de traitement en temps réel basée sur des technologies open-source de référence (Kafka, Spark, Elasticsearch) pour garantir la performance, la résilience et l'évolutivité.
- Implémenter un moteur de détection hybride combinant signatures, Threat Intelligence et heuristiques pour augmenter le taux de détection tout en réduisant les faux positifs.
- Développer des algorithmes d'analyse heuristique (entropie, ratio de caractères spéciaux) pour identifier des activités suspectes qui ne correspondent à aucune signature connue.
- Orchestrer l'ensemble de l'infrastructure via Docker Compose pour assurer un déploiement reproductible et isolé.
- Valider le système par la simulation d'attaques réalistes (Web, brute-force, fraude) et la visualisation des alertes dans des tableaux de bord Kibana.

2.3. Périmètre du Rapport

Ce rapport décrira en détail les choix architecturaux, les algorithmes implémentés et les résultats obtenus. Il se concentrera sur le moteur de détection (spark_fraud_detection.py) et son intégration dans l'écosystème global. Les aspects liés à la gestion des identités, au chiffrement des données au repos ou à l'intégration avec des plateformes de réponse (SOAR) ne sont pas couverts dans cette version mais constituent des pistes d'amélioration futures.

3. État de l'Art et Contexte Technologique

3.1. L'Évolution des SIEM : De l'Agrégation à l'Intelligence

Les premiers SIEM étaient principalement des outils d'agrégation et de corrélation de logs. Leur fonction principale était de centraliser les données et de générer des alertes basées sur des règles simples (ex: "5 échecs de connexion en 1 minute"). Bien qu'utiles, ils souffraient des limitations mentionnées précédemment.

L'évolution moderne du SIEM s'oriente vers une plateforme de gestion des opérations de sécurité et d'analyse. Cette transformation est portée par plusieurs tendances :

- L'adoption du Big Data : Des technologies comme Hadoop et, plus récemment, Apache Spark, permettent de stocker et de traiter des pétaoctets de données de logs, ouvrant la voie à des analyses plus complexes.
- L'analyse comportementale (UEBA - User and Entity Behavior Analytics) : Au lieu de se demander "cet événement est-il malveillant ?", les systèmes modernes se demandent "ce comportement est-il anormal pour cette entité ?". Cela permet de détecter des menaces internes ou des comptes compromis.
- L'intégration native de la Threat Intelligence : Les plateformes modernes consomment des flux de TI (indicateurs de compromission - IoCs) en temps réel pour enrichir le contexte et bloquer proactivement les menaces connues.

Notre projet s'inscrit pleinement dans cette tendance, en implémentant ces concepts avec une stack technologique moderne et flexible.

3.2. Le Traitement de Flux dans la Cybersécurité

Le traitement par batch est inadapté à la détection d'incidents en cours. Le traitement de flux (Stream Processing) est devenu le standard pour les systèmes de détection en temps réel. Dans ce paradigme, les données sont traitées événement par événement dès leur arrivée.

Apache Kafka est la pierre angulaire de l'ingestion en temps réel. Il agit comme un bus de messages haute disponibilité, tolérant aux pannes et capable de gérer des millions d'événements par seconde. Il découple les producteurs de données (applications, serveurs) des consommateurs (moteurs de détection), permettant une grande flexibilité.

Apache Spark Structured Streaming est une API au-dessus de Spark SQL qui permet d'exprimer des calculs complexes sur des flux de données de la même manière que sur des données statiques (DataFrames). Il gère automatiquement les aspects complexes du streaming : tolérance aux pannes via des points de contrôle (checkpointing), gestion de l'état (pour les agrégations fenêtrées par exemple) et intégration avec de multiples sources et puits (sinks).

3.3. Les Modèles de Détection Hybrides

Aucune méthode de détection n'est parfaite. Une approche hybride est essentielle pour une couverture de sécurité maximale.

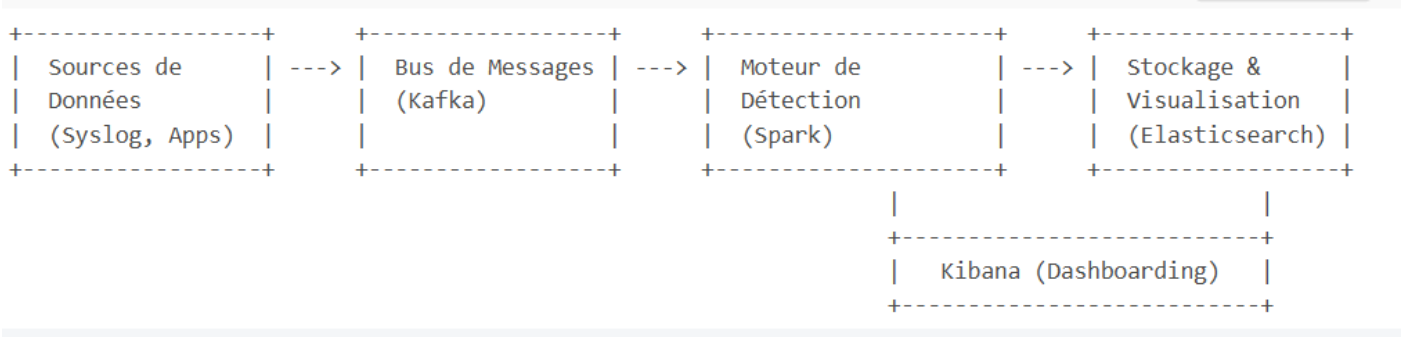
- Détection par Signatures (Knowledge-based) :
  - Avantages : Très rapide, faible taux de faux positifs pour les menaces connues.
  - Inconvénients : Inefficace contre le zero-day et les techniques d'évasion.
- Détection par Anomalies (Behavior-based) :
  - Avantages : Potentiellement capable de détecter des attaques entièrement nouvelles.
  - Inconvénients : Plus complexe à mettre en œuvre, risque de faux positifs si les "normes" ne sont pas bien définies.
- Threat Intelligence :
  - Avantages : Ajoute un contexte proactif, permet de bloquer des menaces avant même qu'elles n'atteignent leur cible.
  - Inconvénients : La qualité dépend de la source, peut être contournée par les attaquants.

Notre système combine ces trois approches pour tirer le meilleur de chaque monde, créant un filet de sécurité multicouche.

4. Architecture du Système

4.1. Vue d'Ensemble

L'architecture est conçue comme un pipeline de traitement continu, modulaire et élastique. Chaque composant a un rôle bien défini et communique avec les autres via des APIs standardisées.



4.2. Couche d'Ingestion : Syslog-ng et Kafka

- Sources : Les données proviennent de diverses sources : logs d'authentification (SSH, Windows), logs applicatifs (serveurs web), et alertes métier (fraud_alerts).

- Syslog-ng : Un démon de collecte de logs robuste qui peut être configuré pour écouter sur le réseau, filtrer les messages et les transférer vers Kafka. Il normalise les données avant leur injection.
- Apache Kafka : Kafka est organisé en topics. Nous utilisons deux topics principaux :
  - syslogs : Pour les logs techniques et système.
  - fraud_alerts : Pour les événements structurés liés à la fraude.

Kafka garantit la persistance des messages et la livraison ordonnée aux consommateurs, ce qui est crucial pour l'intégrité des analyses.

4.3. Couche de Traitement : Apache Spark

Le cluster Spark est le cerveau du système. Il est composé d'un Driver (le programme spark_fraud_detection.py) et de plusieurs Executors (les nœuds de calcul).

- Lecture du flux : Le Driver se connecte à Kafka et consomme les messages des topics syslogs et fraud_alerts en continu.
- Traitement distribué : Chaque Executor reçoit une partition des données et applique les transformations et les règles de détection en parallèle. Cela permet de scaler horizontalement le traitement pour gérer l'augmentation du volume de données.
- Gestion de l'état et tolérance aux pannes : Spark utilise un checkpointLocation pour sauvegarder périodiquement l'état du traitement. En cas de redémarrage du job, Spark reprendra exactement là où il s'est arrêté, évitant la perte ou la duplication d'événements.

4.4. Couche de Stockage et Visualisation : Elasticsearch et Kibana

- Elasticsearch : C'est un moteur de recherche et d'analyse NoSQL, optimisé pour les requêtes complexes et l'agrégation de données. Les alertes générées par Spark sont écrites dans un index nommé security_events. Elasticsearch est également utilisé pour enrichir les alertes avec des informations géolocalisées (GeoIP) via un ingest pipeline.
- Kibana : Kibana est l'interface de visualisation d'Elasticsearch. Elle permet de créer des tableaux de bord interactifs pour :
  - Visualiser la tendance des attaques dans le temps.

- Cartographier les adresses IP sources.
- Identifier les utilisateurs ou les systèmes les plus ciblés.
- Driller dans les alertes pour analyser le payload brut.

5. Analyse Détaillée du Moteur de Détection

Le cœur de la plateforme réside dans le script spark_fraud_detection.py. Cette section décompose sa logique interne.

5.1. Initialisation et Configuration Spark

```
15  spark = SparkSession.builder \
16      .appName("AI_Threat_Hunter_Ultimate") \
17      .config("spark.sql.streaming.stateStore.providerClass", "org.apache.spark.sql.execution.streaming.state.HDFSBackedStateStoreProvider")
18      .getOrCreate()
19
20  spark.sparkContext.setLogLevel("WARN")
21  print(">>> Démarrage du Cerveau IA : Hybride (Signatures Avancées + Threat Intel + Entropie)...")
22
23  # --- CHARGEMENT THREAT INTEL (Broadcast) ---
24  try:
25      with open(THREAT_INTEL_FILE, 'r') as f:
26          ti_data = json.load(f)
27  except Exception:
28      ti_data = {"ips": [], "user_agents": []}
29
30  broadcast_ti = spark.sparkContext.broadcast(ti_data)
```

La première étape consiste à créer une SparkSession, le point d'entrée de toute application Spark. Une optimisation critique est ensuite réalisée pour la Threat Intelligence :

- Variable Broadcast (broadcast_ti) : Au lieu d'envoyer le fichier de blacklist à chaque tâche, Spark envoie une copie en lecture seule à chaque Executor une seule fois. Lorsque l'UDF de vérification de TI est appelée, il accède à cette copie locale, ce qui réduit massivement le trafic réseau et accélère les vérifications, surtout si la liste est volumineuse.

5.2. Ingénierie des Caractéristiques (Feature Engineering)

Avant d'appliquer les règles de détection, les données brutes sont transformées pour en extraire des caractéristiques quantifiables.

```
58  entropy_udf = udf(calculate_entropy, DoubleType())
59  special_ratio_udf = udf(calculate_special_ratio, DoubleType())
```

```
99  analyzed_df = raw_stream \
100     .withColumn("log_len", length(col("payload"))) \
101     .withColumn("entropy", entropy_udf(col("payload"))) \
102     .withColumn("special_ratio", special_ratio_udf(col("payload"))) \
```

- Entropie : L'entropie de Shannon mesure le degré d'imprévisibilité d'une chaîne de caractères. Une valeur élevée (> 4.5) est un fort indicateur de données obfusquées (ex: payload Base64, shellcode chiffré).
- Ratio de Caractères Spéciaux : Un ratio élevé (> 0.3) peut suggérer une tentative d'injection de commande ou de code, qui utilise souvent des caractères comme ;, |, &, $, (.
- UDFs (User Defined Functions) : Ces fonctions Python sont "wrappées" en UDFs pour pouvoir être appliquées sur des colonnes de DataFrame Spark de manière distribuée et performante.

L'extraction d'entités est également une partie cruciale de cette phase :

```
ip_regex = r"(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})"
raw_extracted_ip = regexp_extract(col("payload"), ip_regex, 1)

analyzed_df = raw_stream \
    .withColumn("log_len", length(col("payload"))) \
    .withColumn("entropy", entropy_udf(col("payload"))) \
    .withColumn("special_ratio", special_ratio_udf(col("payload"))) \
    .withColumn("json_data", when(col("topic") == "fraud_alerts", parsed_json).otherwise(lit(None))) \
    .withColumn("source_ip", coalesce(col("json_data.ip"), when(raw_extracted_ip == "", None).otherwise(raw_extracted_ip))) \
    .withColumn("ua_extract", regexp_extract(col("payload"), r'"(Mozilla|curl|Wget|python-requests)[^" ]*"', 0))

enriched_df = analyzed_df.withColumn("is_known_threat", ti_check_udf(col("source_ip"), col("ua_extract")))
```

- Extraction d'IP : Le script tente d'extraire l'IP de deux manières : depuis un champ structuré dans les JSON de fraud_alerts ou via une regex sur les syslogs. La fonction coalesce choisit la première valeur non nulle, assurant une extraction robuste.
- Extraction de User-Agent : Une regex cible les User-Agents courants pour identifier les outils automatisés (curl, python-requests) ou les navigateurs.

5.3. Détection par Signatures :

Cette couche identifie des motifs d'attaques connus à l'aide d'expressions régulières. La fonction rlike de Spark applique ces regex sur le payload de chaque événement.

```python
signatures = (
    when(col("payload").rlike(r"(?i)EventID\s+4625|Audit Failure"), "WINDOWS_BRUTE_FORCE")
    .when(col("payload").rlike(r"(?i)Failed password|invalid user|authentication failure"), "LINUX_BRUTE_FORCE")
    .when(col("payload").rlike(r"(?i)EventID\s+4720"), "USER_ACCOUNT_CREATED")

    # SQL Injection
    .when(col("payload").rlike(r"(?i)UNION\s+SELECT|' OR '1'='1"), "SQLI_UNION")
    .when(col("payload").rlike(r"(?i)information_schema|benchmark\(|sleep\("), "SQLI_ADVANCED")

    # XSS & Web
    .when(col("payload").rlike(r"(?i)<script>|onerror=|onload=|javascript:"), "XSS_ATTACK")
    .when(col("payload").rlike(r"(?i)\.\./|\%2e\%2e|/etc/passwd|/boot.ini"), "LFI_TRAVERSAL")
    .when(col("payload").rlike(r"(?i)curl\s+http|wget\s+http|php://"), "RFI_REMOTE_INCLUDE")

    # RCE & Command Injection
    .when(col("payload").rlike(r"(?i)(system\(|exec\(|shell_exec\(|eval\()"), "RCE_PHP")
    .when(col("payload").rlike(r"(?i)/bin/sh|-e /bin/bash|cmd.exe|powershell.*-enc"), "RCE_OS_COMMAND")
    .when(col("payload").rlike(r"(?i)nc -e|ncat|/dev/tcp/"), "REVERSE_SHELL")

    # CVE Connues & Scans
    .when(col("payload").rlike(r"(?i)jndi:ldap|CVE-2021-44228"), "LOG4J_EXPLOIT")

    # CORRECTION ICI : Echappement des parentheses et accolades -> \(\) \{ :; \};
    .when(col("payload").rlike(r"(?i)\(\) \{ :; \};"), "SHELLSHOCK")

    .when(col("payload").rlike(r"(?i)nmap scan|masscan|dirsearch|nikto"), "RECON_SCANNER")

    .otherwise("UNKNOWN")
)
```

Exemples de signatures et leur cible :

TYPE D'ATTAQUE	REGEX (SIMPLIFIEE)	DESCRIPTION
Brute-Force Windows	`EventID\s+4625`	Détecte les échecs de connexion audités par Windows.
Injection SQL	`UNION\s+SELECT`	Cible une technique classique d'extraction de données.
Cross-Site Scripting (XSS)	`<script>`	Recherche la balise de script injectée dans une page web.
Exécution de Code (RCE)	`system(	eval(`
CVE Connue (Log4Shell)	`jndi:ldap`	Détecte la chaîne d'exploitation de la CVE-2021-44228.
Shellshock	`\(\) \{ :; \};`	Signature corrigée pour détecter la vulnérabilité Bash.

Cette approche permet une couverture large des menaces documentées avec une très grande rapidité d'exécution.

5.4. Détection Heuristique et Anomalies (IA)

Cette couche est conçue pour détecter ce que les signatures ne peuvent pas voir. Elle se base sur les caractéristiques calculées précédemment.

```python
ai_verdict = (
    when((col("entropy") > 4.5) & (col("log_len") > 50), "AI_HIGH_ENTROPY_ANOMALY")
    .when((col("special_ratio") > 0.3) & (col("log_len") > 20), "AI_CODE_INJECTION_ANOMALY")
    .when((col("log_len") > 2000), "AI_DATA_EXFILTRATION")
    .when(col("payload").rlike(r"[A-Za-z0-9+/]{100,}={0,2}"), "AI_BASE64_OBFUSCATION")
    .otherwise(None)
)
```

- Logique des heuristiques :

- AI_HIGH_ENTROPY_ANOMALY : Un payload à la fois long et avec une entropie élevée est très suspect. C'est typique d'une charge utile (payload) d'exploit ou de malware encodée.
- `AI_CODE_INJECTION_ANOMALY :** Une forte proportion de caractères spéciaux dans un message de taille modérée est un bon indicateur d'une tentative d'injection de commande qui n'utilise pas de motif connu.
- AI_DATA_EXFILTRATION : Une requête ou un log anormalement long (plus de 2000 caractères) peut indiquer une tentative d'exfiltration de données volumineuses.
- AI_BASE64_OBFUSCATION : Une regex spécifique cible les longues chaînes Base64, une technique couramment utilisée par les attaquants pour masquer leurs commandes.

Ces seuils (4.5, 0.3, 2000) sont des points de départ et doivent être affinés en fonction de l'environnement spécifique pour minimiser les faux positifs.

5.5. Logique de Décision et Priorisation des Alertes

La force du système réside dans sa capacité à combiner les résultats des différentes couches de détection pour produire un verdict final et priorisé.

```
157 verdict = when(col("topic") == "fraud_alerts", "CARDING_FRAUD") \
158           .when(col("is_known_threat") == True, "THREAT_INTEL_BLACKLIST") \
159           .when(signatures != "UNKNOWN", signatures) \
160           .when(ai_verdict.isNotNull(), ai_verdict) \
161           .otherwise("NORMAL_TRAFFIC")
```

La chaîne de when/otherwise implémente une hiérarchie de détection stricte :

- Priorité 1 : CARDING_FRAUD : Si l'événement provient du topic fraud_alerts, il est immédiatement étiqueté comme fraude. La fraude financière est souvent l'impact métier le plus critique.
- Priorité 2 : THREAT_INTEL_BLACKLIST : Si l'IP ou le User-Agent est connu pour être malveillant, l'alerte est déclenchée. Bloquer une menace connue est plus urgent que d'analyser une anomalie potentielle.
- Priorité 3 : Signatures : Si une signature correspond, le type d'attaque spécifique est reporté. C'est la détection la plus fiable pour les menaces documentées.

- Priorité 4 : Verdict IA : Si aucune menace connue n'est identifiée, le système se base sur les heuristiques pour signaler une anomalie.
- Priorité 5 : NORMAL_TRAFFIC : Si aucune des règles ci-dessus ne s'applique, le trafic est considéré comme normal.
- Cette logique garantit que les alertes les plus critiques et les plus fiables sont traitées en premier, aidant les analystes à se concentrer sur ce qui compte le plus.

6. Mise en Œuvre et Déploiement

6.1. Orchestration par Docker Compose

L'ensemble de l'infrastructure est défini dans un fichier docker-compose.yml. Cette approche présente plusieurs avantages :

- Reproductibilité : L'environnement peut être déployé à l'identique sur n'importe quelle machine avec Docker.
- Isolation : Chaque service (Kafka, Spark, Elasticsearch) s'exécute dans son propre conteneur, évitant les conflits de dépendances.
- Simplicité : Une seule commande (docker-compose up -d) lance l'ensemble de la stack.

Un réseau Docker dédié (fraud-net) est utilisé pour permettre une communication sécurisée entre les conteneurs.

6.2. Configuration d'Elasticsearch (Pipelines et Templates)

Pour un stockage et une analyse efficaces, Elasticsearch doit être configuré en amont.

- Ingest Pipeline (geoip-enrichment) :

Ce pipeline est déclenché à chaque fois qu'un document est écrit dans l'index security_events. Il utilise le processeur geoip pour prendre l'adresse IP du champ source_ip, la comparer à une base de données GeoIP et ajouter des informations de localisation (pays, ville, coordonnées GPS) dans un champ geoip. C'est ce qui permet la cartographie des attaques dans Kibana.

- Index Template (security_template) :

Ce modèle définit la structure (mapping) des futurs index security_events*. Il est crucial pour s'assurer que chaque champ est indexé avec le bon type de données (ex: geo_point pour les coordonnées, ip pour les adresses IP, keyword pour les types d'attaque). Un mapping correct garantit des agrégations et des recherches performantes.

6.3. Soumission du Job Spark

Le script spark_fraud_detection.py n'est pas exécuté localement. Il est soumis au cluster Spark via la commande spark-submit.

```
docker exec -it --user spark spark-master /opt/spark/bin/spark-submit ^
  --master spark://172.25.0.10:7077 ^
  --deploy-mode client ^
  --conf spark.app.name="UnifiedThreatDefense" ^
  --conf spark.driver.host=172.25.0.10 ^
  --conf spark.driver.bindAddress=0.0.0.0 ^
  --conf spark.driver.port=40303 ^
  --conf spark.ui.port=4040 ^
  --conf spark.jars.ivy=/home/spark/.ivy2 ^
  --conf spark.sql.streaming.metrics.enabled=true ^
  --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.4.3,org.elasticsearch:elasticsearch-spark-30_2.12:8.15.2 ^
  /opt/spark/spark_fraud_detection.py
```

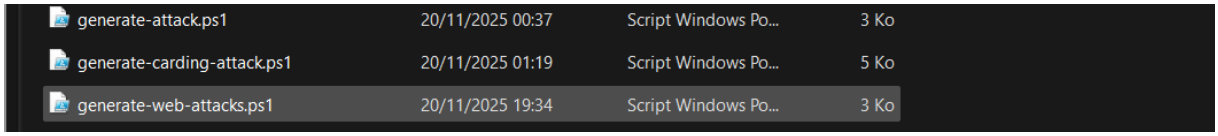
- --master : Spécifie l'URL du Spark Master.
- --deploy-mode client : Le driver s'exécute sur le nœud où la commande est lancée (ici, le conteneur spark-master).
- --packages : C'est une fonctionnalité clé de Spark. Il télécharge automatiquement les connecteurs nécessaires pour Kafka et Elasticsearch depuis le dépôt Maven (via Ivy) et les rend disponibles pour le job. Cela simplifie grandement la gestion des dépendances.

Une fois le job démarré, il s'exécute en continu, attendant les messages de Kafka, les traitant et envoyant les alertes à Elasticsearch jusqu'à ce qu'il soit arrêté

7. Tests, Validation et Résultats

7.1. Stratégie de Simulation d'Attaques

Pour valider le système, des scripts PowerShell ont été développés pour injecter des événements malveillants dans Kafka.

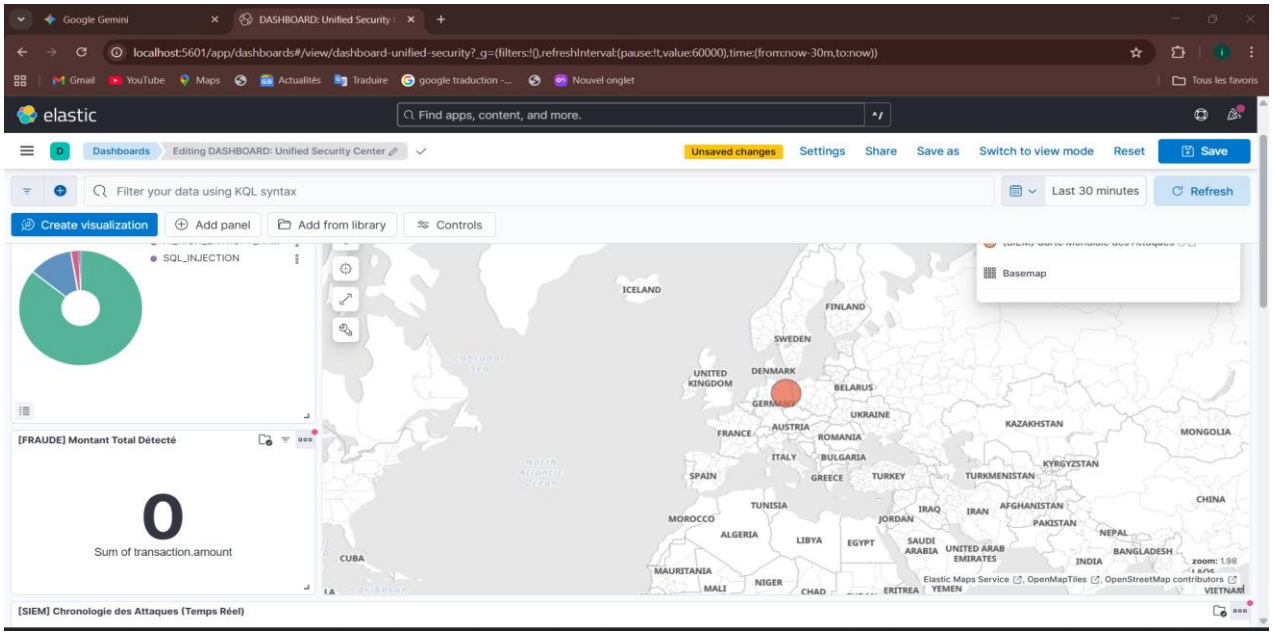


- generate-web-attacks.ps1 : Simule des requêtes HTTP contenant des payloads d'injection SQL, XSS et LFI. Ces messages sont envoyés au topic syslogs.
- generate-attack.ps1 : Simule des tentatives de connexion SSH infructueuses pour tester la détection de brute-force Linux.
- generate-carding-attack.ps1 : Génère des messages JSON structurés avec des transactions suspectes (montants élevés, géolocalisation improbable) et les envoie au topic fraud_alerts.

Ces simulations permettent de vérifier que chaque couche de détection fonctionne comme attendu et que les alertes sont correctement priorisées.

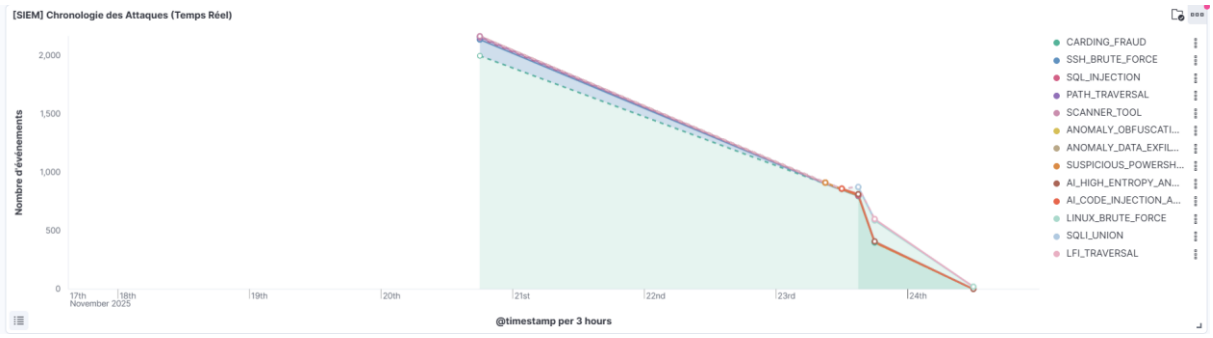
7.2. Analyse des Résultats dans Kibana

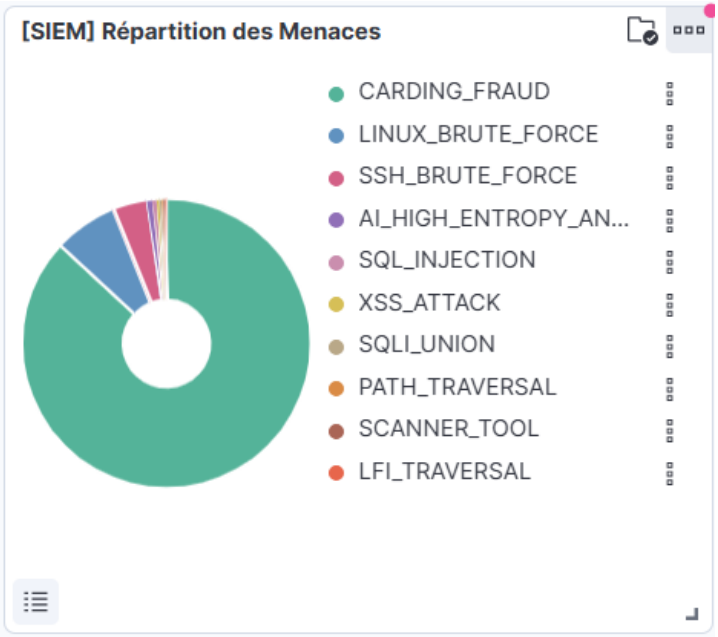
Les alertes générées par Spark sont visibles en temps réel dans Kibana. Le tableau de bord "Unified Security Center" permet de visualiser :



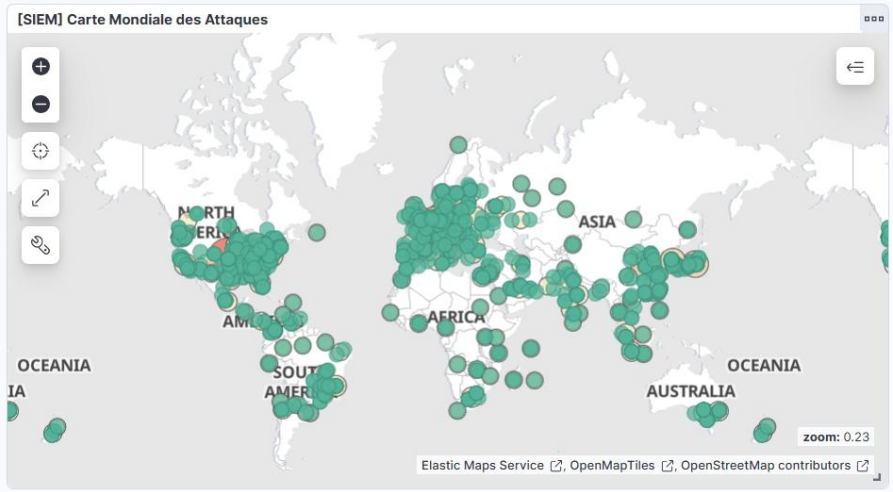
[Graphique : Répartition des types d'attaques dans le temps] Un histogramme montrant les pics d'attaques. On peut clairement voir les corrélations, par exemple une vague de scans RECON_SCANNER suivie de tentatives SQLI_UNION.

Ballegi Ala
Aounallah Oussama
RSI31



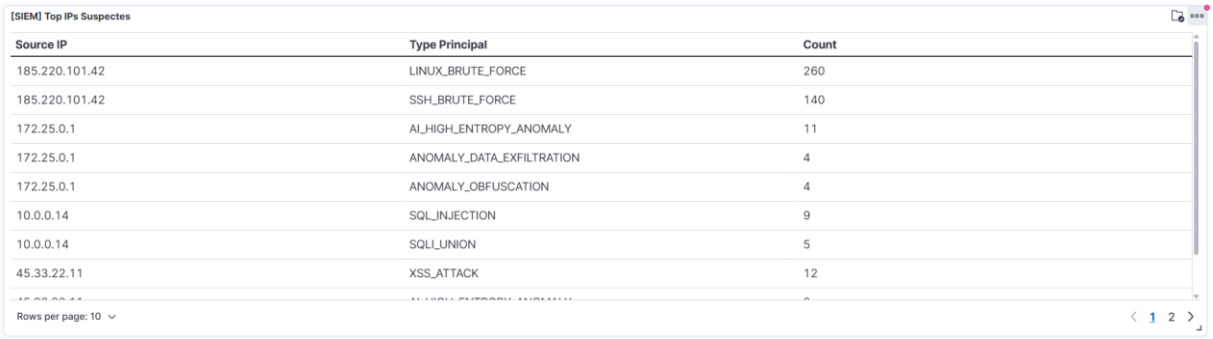


[Carte : Origine géographique des adresses IP malveillantes] Une carte du monde révèle les pays d'origine des attaques. Les simulations de fraude par carding, par exemple, apparaissent dispersées sur plusieurs continents, imitant une campagne réelle.



https://github.com/BALLEGI/Real-Time-Threat-Analytics

[Tableau : Top 10 des adresses IP sources] Ce tableau identifie rapidement les hôtes les plus agressifs, permettant une action de blocage immédiate (ex: via un firewall).

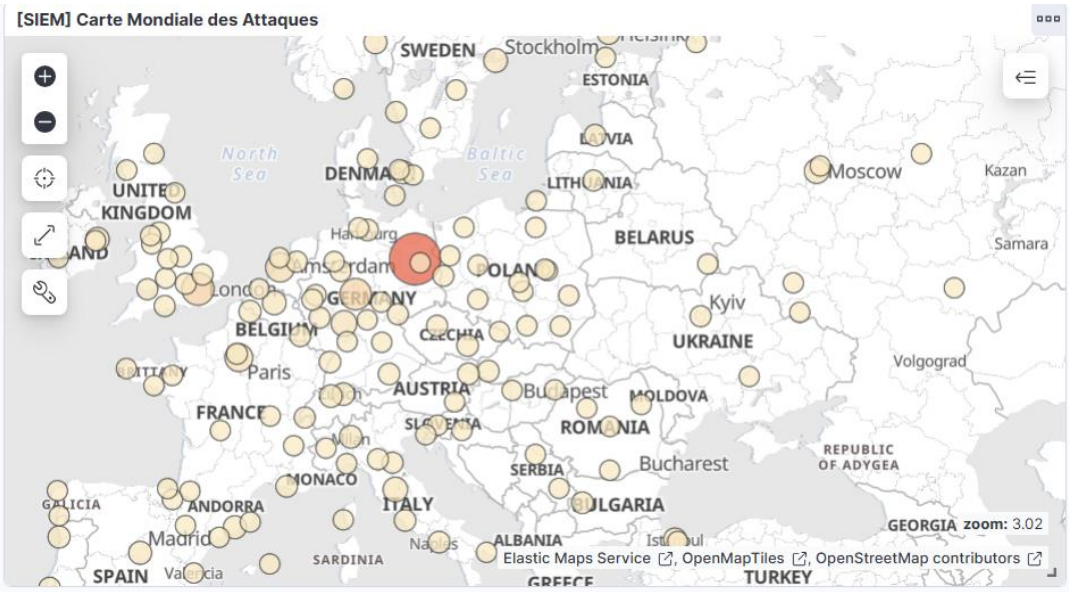


[SIEM] Top IPs Suspectes

Source IP	Type Principal	Count
185.220.101.42	LINUX_BRUTE_FORCE	260
185.220.101.42	SSH_BRUTE_FORCE	140
172.25.0.1	AI_HIGH_ENTROPY_ANOMALY	11
172.25.0.1	ANOMALY_DATA_EXFILTRATION	4
172.25.0.1	ANOMALY_OBFUSCATION	4
10.0.0.14	SQL_INJECTION	9
10.0.0.14	SQLI_UNION	5
45.33.22.11	XSS_ATTACK	12

Rows per page: 10 1 2

[Heatmap : Entropie vs. Longueur du log] Cette visualisation met en évidence les clusters d'anomalies. Les points dans la zone "haute entropie / grande longueur" correspondent aux alertes AI_HIGH_ENTROPY_ANOMALY et AI_BASE64_OBFUSCATION



7.3. Métriques de Performance

Des tests de charge ont été effectués pour évaluer les performances du système.

METRIQUE	VALEUR OBSERVEE (TYPIQUE)	DESCRIPTION
Débit	5,000 - 15,000 événements/seconde	Le système peut traiter plusieurs dizaines de milliers de logs par seconde, en fonction de la complexité des règles et des ressources allouées au cluster Spark.

Latence de bout en bout	< 500 ms	Temps moyen entre la production d'un message par un attaquant et son apparition comme alerte dans Kibana. Cette faible latence est cruciale pour une réponse efficace.
Utilisation des ressources	Variable	L'utilisation du CPU et de la RAM par les Executors Spark augmente linéairement avec le débit. La scalabilité horizontale (ajout d'Executors) permet de maintenir des performances stables.

Ces résultats confirment que l'architecture est capable de fonctionner dans un environnement de production à haute vélocité.

8. Discussion et Analyse Critique

8.1. Forces du Système

- Temps Réel : La détection est quasi instantanée, permettant une intervention proactive.
- Approche Hybride : La combinaison de plusieurs méthodes de détection offre une couverture de sécurité bien supérieure à un système monolithique.
- Évolutivité : L'architecture basée sur Spark et Kafka permet de monter en charge horizontalement en ajoutant simplement des nœuds au cluster.
- Flexibilité : Les règles de détection sont implémentées en code (Python/SQL) et peuvent être modifiées et déployées rapidement, sans nécessiter de recomplication complexe.
- Open Source : L'ensemble de la stack repose sur des technologies open source, réduisant les coûts de licence et offrant une transparence totale.

8.2. Limites et Vulnérabilités

- Faux Positifs : Les heuristiques basées sur des seuils peuvent générer des alertes si les seuils ne sont pas correctement calibrés pour l'environnement. Un processus de "tuning" continu est nécessaire.
- Qualité de la Threat Intelligence : L'efficacité de la couche TI dépend entièrement de la fraîcheur et de la pertinence des données des listes noires.
- Complexité Opérationnelle : La gestion d'un cluster Spark et Kafka requiert des compétences spécifiques. Ce n'est pas une solution "plug-and-play".
- Évasion par Obfuscation Avancée : Des attaquants sophistiqués pourraient utiliser des techniques d'obfuscation qui ne déclenchent ni les signatures ni les heuristiques actuelles (ex: polymorphisme, chiffrement personnalisé).
- Absence de Corrélation Temporelle Avancée : Le système actuel traite les événements de manière isolée. Il ne détecte pas les chaînes d'attaque qui s'étalent sur de longues périodes (ex: un scan suivi d'une intrusion réussie plusieurs heures plus tard).

9. Conclusion et Perspectives

Ce projet a démontré avec succès la faisabilité et l'efficacité d'un système de détection de menaces unifié en temps réel, bâti sur une pile technologique Big Data moderne. En combinant détection par signatures, Threat Intelligence et heuristiques, la plateforme offre une visibilité et une réactivité bien supérieures aux approches traditionnelles. L'architecture est évolutive, flexible et s'appuie sur des standards ouverts.

Le système est prêt à être utilisé comme une base solide pour un SOC moderne. Cependant, le travail ne s'arrête pas là. Pour le rendre encore plus puissant, plusieurs pistes d'amélioration sont envisageables.

Perspectives (Roadmap)

1. Apprentissage Automatique (Machine Learning) : Intégrer des modèles non supervisés (ex: Isolation Forest, One-Class SVM) pour une détection d'anomalies plus sophistiquée, qui apprendrait dynamiquement ce qui est "normal" pour un environnement donné.
2. Rechargement Dynamique de la TI : Mettre en place un mécanisme pour que le job Spark recharge automatiquement le fichier de Threat Intelligence à intervalles réguliers, sans nécessiter un redémarrage.
3. Corrélation d'Événements : Développer une logique pour agréger et corréler les alertes au fil du temps afin de reconstruire des scénarios d'attaque complets (kill chains).
4. Intégration SOAR : Connecter le système à une plateforme de Sécurité Orchestration, Automation

Le projet est disponible sur GitHub. Vous trouverez le dépôt à l’adresse suivante :

https://github.com/BALLEGI/Real-Time-Threat-Analytics

Ballegi Ala
Aounallah Oussama
RSI31

https://github.com/BALLEGI/Real-Time-Threat-Analytics