



Rapport de stage de fin d'études

Présenté en vue de l'obtention du diplôme de Licence en Ingénierie des Systèmes
Informatiques (Computer Engineering)

Spécialité : Systèmes Embarqués et Internet des Objets

Sentinel hive

Détection d'intrusion Wi-Fi basée sur Raspberry Pi, MQTT et Wazuh

Réalisé par :
mohamed slim brahem

Encadrant académique : Mr mohamed elouni

Co encadrant académique : Dr.Ing bayrem triki

Encadrant professionnel : Mr abdelhak achour



Année universitaire : 2025/2026

Page de signatures

Cette page est réservée aux signatures des encadrants ayant accompagné la réalisation de ce projet de fin d'études.

Encadrant académique	Co-encadrant académique	Encadrant professionnel
Mr Mohamed Elouni	Dr. Ing. Bayrem Triki	Mr Abdelhak Achour

Signature	Signature	Signature
-----------	-----------	-----------

Dédicace

À mes parents,

qui n'ont jamais cessé de croire en moi — même quand moi-même j'en
doutais.

Tout ce que je suis, je vous le dois.

À toute ma famille,

pour la chaleur des moments partagés et pour avoir rendu
les instants difficiles plus légers à porter.

À mes encadrants,

Dr. Ing. Bayrem Triki et Mr. Mohamed Ouni,

pour leur disponibilité, leurs conseils éclairés et leur accompagnement
tout au long de ce travail.

Leur exigence bienveillante a été déterminante dans la réussite de ce
projet.

À mon ami Izdin,

et à tous ceux qui ont partagé cette aventure à mes côtés —
pour les fous rires, les discussions qui s'éternisent
et ce soutien sincère qui compte plus qu'ils ne le savent.

*À tous ceux qui ont croisé ma route et laissé quelque chose —
une leçon, un mot, une présence.*

Ce travail porte un peu de chacun d'entre vous.

Remerciements

Ce projet a été une aventure — techniquement exigeante, parfois frustrante, mais profondément enrichissante. Il n'aurait pas pris cette forme sans les personnes qui m'ont accompagné, de près ou de loin, tout au long de cette période.

Je remercie **Mr Mohamed Elouni**, mon encadrant académique, pour sa disponibilité, sa patience et ses retours toujours justes. Il a su me recadrer quand c'était nécessaire, et m'encourager quand ça l'était tout autant.

Merci à **Dr. Ing. Bayrem Triki**, co-encadrant académique, pour ses éclairages techniques et pour avoir posé les bonnes questions au bon moment — celles qui obligent à creuser plus loin que la surface.

Un remerciement sincère à **Mr Abdelhak Achour**, mon encadrant chez Whitecape Technologies, pour m'avoir fait confiance dès le début et pour avoir rendu ce stage concret, stimulant et formateur. Son expérience du terrain a été une vraie valeur ajoutée.

Merci à toute l'**équipe de Whitecape Technologies** pour l'accueil, la bonne ambiance et ce cadre de travail qui donne envie de s'investir.

Merci enfin aux **membres du jury** pour le temps qu'ils consacrent à l'évaluation de ce travail.

Et merci à tous ceux que je n'ai pas cités mais qui savent très bien qu'ils auraient dû l'être.

Table des matières

Remerciements	3
Liste des abréviations	11
1 Étude du projet	12
1.1 Introduction	12
1.2 Contexte général du projet	12
1.3 Organisme d'accueil	13
1.3.1 Présentation de la société	13
1.3.2 Services proposés	14
1.4 Problématiques du projet	15
1.4.1 Visibilité radio limitée	15
1.4.2 Contraintes de couverture multi-canal	15
1.4.3 Fiabilité de la chaîne de supervision	16
1.5 Objectifs du projet	16
1.5.1 Améliorer la visibilité radio du système	16
1.5.2 Structurer une chaîne de collecte reproductible	17
1.5.3 Assurer une corrélation efficace dans le SIEM	17
1.6 Solution proposée	17
1.7 Choix de la méthodologie	18
1.7.1 Les méthodes agiles	18
1.7.2 Choix de la méthode Scrum	20

1.7.3 Présentation de Scrum	21
Conclusion du chapitre	23
2 État de l'art	24
2.1 Introduction	24
2.2 Définition d'un système de détection Wi-Fi distribué	24
2.3 Techniques de détection d'attaques Wi-Fi	25
2.3.1 Détection par signatures	25
2.3.2 Détection par règles et corrélation	25
2.3.3 Détection par analyse comportementale	26
2.3.4 Capture passive en mode moniteur	26
2.3.5 Agrégation temporelle et interprétation	26
2.4 Technologies de détection et de supervision utilisées	26
2.4.1 Raspberry Pi	27
2.4.2 Scapy	27
2.4.3 MQTT	27
2.4.4 Mosquitto	27
2.4.5 Wazuh	27
2.5 Domaines d'application des systèmes de détection Wi-Fi	28
2.5.1 Maison connectée	28
2.5.2 PME et ateliers connectés	28
2.5.3 Laboratoires pédagogiques	28
2.5.4 Événementiel et infrastructures temporaires	29
2.5.5 Analyse et justification du choix technologique	29
2.6 Architecture générale de la solution proposée	29
2.6.1 Vue d'ensemble	29
2.6.2 Couche acquisition : nœuds de capture	30
2.6.3 Couche traitement local : agrégation des trames	31
2.6.4 Couche communication : transport des mesures	31

2.6.5	Couche journalisation : broker Mosquitto	31
2.6.6	Couche supervision : décodage et corrélation	31
2.6.7	Couche visualisation : tableau de bord et analyse	31
2.6.8	Couche monitoring système : plateforme web dédiée	31
2.7	Étude de marché des solutions existantes	32
2.7.1	Snort : solution open source de type NIDS	32
2.7.2	Fingbox : solution orientée grand public	33
2.7.3	Armis : plateforme entreprise agentless	33
2.7.4	Comparaison synthétique et justification	34
2.8	Conclusion	35
3	Spécification des besoins de sécurité et plan de validation	36
3.1	Introduction	36
3.2	Outils et technologies utilisés	36
3.3	Analyse des besoins de sécurité	40
3.3.1	Protection de l'infrastructure de capture	40
3.3.2	Intégrité des événements remontés	40
3.3.3	Disponibilité de la chaîne de supervision	40
3.3.4	Confidentialité et maîtrise des accès	40
3.3.5	Traçabilité et auditabilité	41
3.4	Exigences de sécurité du système	41
3.4.1	Exigences fonctionnelles	41
3.4.2	Exigences non fonctionnelles	42
3.4.3	Hierarchisation des exigences	42
3.5	Architecture de sécurité proposée	43
3.5.1	Vue d'ensemble	43
3.5.2	Sécurisation des nœuds Sentinel	43
3.5.3	Sécurisation de la communication MQTT	43
3.5.4	Journalisation centralisée et corrélation	43

3.5.5	Segmentation logique de l'architecture	44
3.6	Diagrammes UML du système	44
3.6.1	Sprint 1 : analyse fonctionnelle et cadrage	44
3.6.2	Sprint 2 : conception technique et structuration du prototype	48
3.7	Politique de sécurité du prototype	52
3.7.1	Gestion des accès	52
3.7.2	Durcissement des équipements	52
3.7.3	Gestion des journaux et de la conservation des données	52
3.7.4	Réaction en cas d'incident	52
3.7.5	Bonnes pratiques d'exploitation	53
3.8	Plan de validation et de tests de sécurité	53
3.8.1	Objectifs du plan de test	53
3.8.2	Stratégie générale de validation	53
3.8.3	Jeux de tests proposés	54
3.8.4	Critères d'acceptation	54
3.8.5	Limites de la validation	55
3.9	Conclusion	55
4	Environnements de déploiement et scénario d'exécution	56
4.1	Introduction	56
4.2	Sprint 3 : déploiement de l'environnement technique	56
4.2.1	Présentation des environnements techniques	58
4.2.2	Environnement capteur	58
4.2.3	Environnement central	59
4.2.4	Plateforme web de monitoring personnalisée	60
4.2.5	Architecture backend de monitoring	60
4.2.6	Collecte temps réel et visualisation dynamique	61
4.2.7	Apport de la plateforme à la supervision globale	64
4.2.8	Environnement d'attaque	65

4.3 Sprint 4 : scénario d'exécution et validation expérimentale	66
4.3.1 Déroulement du scénario expérimental	67
4.4 Synthèse	73
5 Tests, validation et évaluation de la sécurité	74
5.1 Introduction	74
5.2 Environnement et outillage de test	74
5.2.1 Infrastructure de test	74
5.2.2 Outillage de test – vue d'ensemble	75
5.3 Tests unitaires	76
5.3.1 Objectif et périmètre	76
5.3.2 Tests unitaires – capture et filtrage Scapy	76
5.3.3 Tests unitaires – agrégation temporelle	76
5.3.4 Tests unitaires – format JSON des événements	76
5.3.5 Tests unitaires – seuils de détection	76
5.3.6 Rapport de couverture fonctionnelle	77
5.4 Tests d'intégration de bout en bout	77
5.4.1 Scénario 1 : Chaîne complète MQTT	77
5.4.2 Scénario 2 : Capture Wi-Fi vers corrélation Wazuh	77
5.4.3 Scénario 3 : Visualisation et cycle de vie de l'alerte	77
5.4.4 Scénario 4 : Mapping MITRE ATT&CK	78
5.5 Tests de performance et de charge	78
5.5.1 Test de charge : ingestion MQTT	78
5.5.2 Test de charge : corrélation Wazuh	78
5.5.3 Test de charge : visualisation Dashboard	79
5.6 Tests de résilience et de disponibilité	79
5.6.1 Méthodologie de résilience	79
5.6.2 Résilience 1 : Arrêt d'un nœud capteur	79
5.6.3 Résilience 2 : Indisponibilité du broker Mosquitto	79

5.6.4 Résilience 3 : Dégradation réseau	79
5.7 Tests de reprise après incident	80
5.7.1 Objectif	80
5.7.2 DR1 : Redémarrage complet de la stack	80
5.7.3 DR2 : Reprise du capteur après interruption	80
5.7.4 DR3 : Conservation des journaux et alertes	80
5.8 Tests de sécurité avancés : Pentest	80
5.8.1 Périmètre et méthodologie	80
5.8.2 Attaque 1 : Désauthentification Wi-Fi contrôlée	81
5.8.3 Attaque 2 : Publication MQTT non autorisée	81
5.8.4 Attaque 3 : Scan de ports et fingerprinting	81
5.8.5 Attaque 4 : Tentative d'accès non autorisé au Dashboard	81
5.8.6 Sécurité des secrets et fichiers de configuration	81
5.9 Tests de la logique de détection : Performance et dérive	82
5.9.1 Performance des seuils statiques	82
5.9.2 Sensibilité aux faux positifs	82
5.9.3 Test de dérive du contexte radio	82
5.10 Tests de scalabilité horizontale	82
5.10.1 Ajout d'un nœud Sentinel	82
5.10.2 Répartition multi-canaux	83
5.10.3 Limites de centralisation	83
5.11 Tests du tableau de bord Wazuh	83
5.12 Synthèse des résultats	84
5.13 Analyse critique et axes d'amélioration	84
5.14 Perspectives d'évolution	85
5.15 Conclusion	86
Conclusion générale	88
Résumé et Abstract	89

Table des figures

1.1 Logo de l'entreprise d'accueil : Whitecape Technologies	14
1.2 Cycle général du framework Scrum.	22
1.3 Planification relative du projet Sentinel Hive par sprints.	23
2.1 Architecture générale de la solution Sentinel Hive.	30
2.2 Plateforme Snort.	33
2.3 Plateforme Fingbox.	33
2.4 Plateforme Armis.	34
3.1 Diagramme de cas d'utilisation de Sentinel Hive.	46
3.2 Diagramme de séquence du processus de détection.	47
3.3 Diagramme de séquence du processus d'authentification et d'accès à la pla- teforme.	48
3.4 Diagramme de classes de la solution.	50
3.5 Diagramme d'activité du système Sentinel Hive.	51
4.1 Environnement capteur : nœud Sentinel Hive en cours d'exécution locale.	58
4.2 Environnement central : serveur Ubuntu exécutant la pile Wazuh et les ser- vices de centralisation des événements Sentinel Hive.	59
4.3 Suivi temps réel de l'utilisation CPU de la machine virtuelle Ubuntu.	62
4.4 Suivi temps réel de l'utilisation de la mémoire RAM.	63
4.5 Suivi temps réel de l'activité réseau de la machine virtuelle Ubuntu.	64
4.6 Environnement d'attaque : poste Kali Linux	65

4.7	État nominal de la plateforme Wazuh avant l'apparition d'une anomalie radio.	68
4.8	Phase de reconnaissance du réseau Wi-Fi cible à l'aide de airodump-ng.	69
4.9	Déclenchement contrôlé d'une attaque de désauthentification sur le réseau	
	Wi-Fi cible.	70
4.10	Corrélation SIEM dans Wazuh et génération d'une alerte de sécurité enrichie.	71
4.11	Enrichissement de l'alerte Sentinel Hive par mapping MITRE ATT&CK.	72
4.12	Visualisation analytique des indicateurs et alertes générés par Sentinel Hive.	72
5.1	Chaîne fonctionnelle de Sentinel Hive, de la capture radio à la corrélation	
	SIEM.	78
5.2	Perspectives d'évolution de Sentinel Hive vers un composant de supervision	
	intégré.	86

Liste des tableaux

1.1 Comparaison des principales méthodes agiles.	20
2.1 Comparaison synthétique des plateformes existantes.	34
3.1 Outils et technologies utilisés dans Sentinel Hive – partie 1.	38
3.2 Outils et technologies utilisés dans Sentinel Hive – partie 2.	39
3.3 Sprint Backlog détaillé du Sprint 1.	45
3.4 Sprint Backlog détaillé du Sprint 2.	49
3.5 Plan synthétique de validation et de tests de sécurité.	54
4.1 Sprint Backlog détaillé du Sprint 3.	57
4.2 Sprint Backlog détaillé du Sprint 4.	67
5.1 Outillage principal utilisé pour les tests de Sentinel Hive.	75
5.2 Synthèse des résultats de tests de Sentinel Hive.	84
5.3 Récapitulatif des améliorations proposées pour Sentinel Hive.	85

Liste des abréviations

Abréviation	Signification
ACK	Acknowledgement
API	Application Programming Interface
CPU	Central Processing Unit
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
IDS	Intrusion Detection System
IoT	Internet of Things
IP	Internet Protocol
IT	Information Technology
JSON	JavaScript Object Notation
MAC	Medium Access Control
MQTT	Message Queuing Telemetry Transport
NIDS	Network Intrusion Detection System
RAM	Random Access Memory
SIEM	Security Information and Event Management
SOAR	Security Orchestration, Automation and Response
SOC	Security Operations Center
TPE	Très petite entreprise
TLS	Transport Layer Security
USB	Universal Serial Bus
VM	Virtual Machine
Wi-Fi	Wireless Fidelity
WIDS	Wireless Intrusion Detection System

Chapitre 1

Étude du projet

1.1 Introduction

Ce premier chapitre présente le cadre général du projet **Sentinel Hive**. Il situe le travail dans son contexte académique et professionnel, expose le besoin à traiter et introduit les principaux choix qui orientent la réalisation du prototype.

Il présente successivement le contexte général du projet, l'organisme d'accueil, les problématiques identifiées, les objectifs visés, la solution proposée ainsi que la méthodologie adoptée. L'objectif est de poser une base claire avant d'aborder, dans les chapitres suivants, l'étude du domaine et les aspects techniques de la solution.

1.2 Contexte général du projet

L'essor des objets connectés a renforcé l'usage du Wi-Fi dans les environnements domestiques, industriels et pédagogiques[9, 8]. Cette évolution facilite le déploiement des équipements, mais elle augmente également l'exposition aux attaques radio visant directement les communications sans fil.

Dans ce contexte, une supervision limitée au trafic IP ou aux journaux classiques ne suffit pas toujours. Certaines attaques, comme les *deauthentication floods*, les *disassociation floods* ou le brouillage, apparaissent d'abord au niveau des trames 802.11[9, 11]. Le

projet **Sentinel Hive** propose donc une architecture légère fondée sur plusieurs nœuds Raspberry Pi[7] de capture passive, avec une centralisation des événements via MQTT[6], Mosquitto[4] et Wazuh[5] afin d'améliorer la visibilité radio et la supervision du réseau.

1.3 Organisme d'accueil

Le projet est réalisé au sein de **Whitecape Technologies**, une société de services informatiques euro-tunisienne spécialisée dans le développement logiciel en mode *nearshore*. L'entreprise accompagne des éditeurs de logiciels et des entreprises dans la conception, le développement et le pilotage de solutions numériques, en combinant expertise technique, conseil IT et gestion de projet.

1.3.1 Présentation de la société

Whitecape Technologies se positionne comme une entreprise de services numériques tournée vers le développement de solutions logicielles sur mesure et l'accompagnement technique de ses clients. Son modèle *nearshore* lui permet de proposer des équipes dédiées, capables d'intervenir sur des projets de développement à moyen et long terme tout en maintenant une proximité organisationnelle avec les besoins métier des clients.

L'entreprise intervient également dans des missions de conseil IT et de gestion de projet, ce qui lui permet d'assurer non seulement la réalisation technique des solutions, mais aussi leur cadrage, leur suivi et leur évolution. Ce cadre professionnel constitue un environnement pertinent pour un projet comme Sentinel Hive, qui mobilise à la fois conception technique, intégration d'outils et structuration d'une solution logicielle cohérente.



FIG. 1.1 – Logo de l'entreprise d'accueil : Whitecape Technologies

1.3.2 Services proposés

D'après son portefeuille de services, Whitecape Technologies propose principalement les prestations suivantes :

- **Développement logiciel en mode nearshore.** Mise à disposition d'équipes dédiées pour des projets de longue durée ou pour des besoins ciblés en compétences techniques ;
- **Développement sur mesure.** Conception et réalisation de solutions adaptées aux besoins spécifiques des clients en s'appuyant sur des technologies récentes ;
- **Gestion du cycle de vie applicatif.** Prise en charge des différentes étapes d'un produit logiciel, depuis la conception jusqu'à la maintenance, au débogage et aux évolutions ;
- **Développement d'applications web et mobiles.** Réalisation d'applications desktop, web et mobiles selon les besoins fonctionnels des entreprises et des éditeurs.

Ces services montrent que l'entreprise évolue dans un environnement fortement orienté ingénierie logicielle, intégration et accompagnement projet, ce qui constitue un cadre cohérent pour le développement d'une solution expérimentale comme Sentinel Hive.

1.4 Problématiques du projet

Le projet répond à une problématique centrale :

Comment concevoir un système de détection d'intrusion Wi-Fi léger, autonome et capable d'offrir une bonne visibilité radio sur plusieurs canaux dans un environnement IoT ?

Cette problématique générale peut être décomposée en plusieurs difficultés techniques et opérationnelles.

1.4.1 Visibilité radio limitée

Dans un environnement Wi-Fi, un grand nombre d'événements significatifs se produisent directement au niveau des trames 802.11[9]. Or, une supervision classique fondée uniquement sur les journaux des équipements ou sur le trafic IP ne permet pas toujours de distinguer des comportements anormaux tels qu'un flood de désauthentification ou une activité excessive de sondage radio[11, 8]. La première difficulté consiste donc à obtenir une visibilité suffisamment fine sur la couche liaison sans perturber le réseau surveillé.

Cette exigence justifie l'utilisation du mode moniteur et d'une capture passive des trames[3, 9]. Une telle approche permet de conserver une observation non intrusive tout en collectant des informations essentielles sur le comportement radio de l'environnement ciblé.

1.4.2 Contraintes de couverture multi-canal

La surveillance d'un seul canal ne suffit pas lorsque l'environnement radio est réparti sur plusieurs fréquences Wi-Fi[9]. Un capteur unique ne peut pas suivre simultanément l'ensemble du spectre utile sans perte de précision, surtout si l'on souhaite détecter des comportements rapides ou intermittents. Il devient alors nécessaire de répartir la capture sur plusieurs nœuds, chacun affecté à un canal donné, conformément au principe des architectures distribuées de surveillance[10, 11].

Cette contrainte introduit un défi d'architecture : il ne s'agit plus seulement de capturer des trames, mais de coordonner plusieurs points d'observation afin d'obtenir une vision plus large, plus stable et plus représentative de l'activité radio globale.

1.4.3 Fiabilité de la chaîne de supervision

La valeur du système ne repose pas uniquement sur la qualité de la capture. Elle dépend aussi de la capacité à transporter, journaliser et corréler les événements sans perte majeure ni ambiguïté. Entre le script Scapy[3], la publication MQTT[6], la journalisation Mosquitto[4] et la corrélation Wazuh[5], chaque maillon doit rester cohérent avec les autres.

Une faiblesse dans cette chaîne peut altérer la qualité des alertes, introduire des retards ou rendre l'analyse plus difficile. Le projet doit donc proposer une solution où la légèreté du capteur s'accompagne d'une structuration rigoureuse de la remontée d'information.

1.5 Objectifs du projet

L'objectif global est de développer une architecture de détection d'intrusion Wi-Fi multi-nœuds, fondée sur la capture passive de trames 802.11 sur plusieurs canaux et sur leur corrélation centralisée dans Wazuh.

1.5.1 Améliorer la visibilité radio du système

Le premier objectif consiste à mettre en œuvre plusieurs nœuds Raspberry Pi[7] de capture passive, chacun affecté à un canal Wi-Fi donné, afin d'observer simultanément plusieurs portions du spectre. Cette organisation doit permettre une meilleure couverture de l'environnement radio et une augmentation de la capacité de détection des phénomènes anormaux.

L'enjeu n'est pas seulement de collecter davantage de données, mais d'améliorer la représentativité des observations produites par le système, en limitant les angles morts liés à une surveillance mono-canal.

1.5.2 Structurer une chaîne de collecte reproductible

Le deuxième objectif est de construire une chaîne de collecte simple, cohérente et reproductible. Les trames capturées doivent être agrégées localement par un script Python/Scapy[3], puis transformées en messages JSON normalisés transmis vers un broker MQTT[6]. Cette étape garantit une séparation claire entre la phase d'acquisition et la phase de supervision.

Une telle structuration facilite les tests, la maintenance du prototype et sa réutilisation dans d'autres contextes expérimentaux. Elle permet également de conserver une architecture lisible, ce qui est essentiel dans un mémoire de fin d'études.

1.5.3 Assurer une corrélation efficace dans le SIEM

Le troisième objectif vise à exploiter les données collectées dans une plateforme de supervision capable de produire des alertes interprétables. Les messages remontés doivent être journalisés, transférés via l'agent Wazuh[5] puis corrélés à l'aide de décodeurs et de règles personnalisés.

L'ambition est de transformer des observations radio brutes en événements de sécurité exploitables, visibles dans un tableau de bord et suffisamment structurés pour soutenir une analyse critique des résultats obtenus.

1.6 Solution proposée

Afin de répondre aux problématiques identifiées, notamment la visibilité radio limitée, les contraintes de couverture multi-canal et la nécessité d'une supervision fiable, une solution adaptée au contexte du projet **Sentinel Hive** a été proposée. La solution repose sur une architecture distribuée permettant la capture, la collecte et le traitement des événements Wi-Fi au sein d'un environnement IoT[7, 3, 6, 4, 5].

Elle vise à garantir un compromis entre légèreté, visibilité et fiabilité, tout en assurant une mise en œuvre simple et adaptable aux contraintes réelles d'installation. L'approche

retenue privilégie :

- une architecture modulaire facilitant l'évolution du système ;
- une capture passive distribuée sur plusieurs nœuds et plusieurs canaux Wi-Fi ;
- une infrastructure légère adaptée à un prototype reproductible et exploitable ;
- une centralisation des données en temps quasi réel via MQTT, Mosquitto et Wazuh.

La conception détaillée de l'architecture matérielle et logicielle ainsi que les choix techniques effectués seront présentés dans les chapitres suivants.

1.7 Choix de la méthodologie

La réussite d'un projet technologique repose non seulement sur les solutions techniques adoptées, mais également sur l'organisation choisie pour encadrer son développement. Une méthodologie adaptée permet d'assurer une gestion efficace du temps, une meilleure coordination des activités et un suivi structuré de l'avancement.

Le projet **Sentinel Hive** combine plusieurs dimensions techniques : configuration de nœuds Raspberry Pi[7], capture passive de trames Wi-Fi[9, 3], transport des données par MQTT[6], journalisation via Mosquitto[4] et corrélation dans Wazuh[5]. Ce type de projet présente une complexité réelle, car les résultats obtenus peuvent évoluer après chaque phase de test et certains paramètres techniques doivent être ajustés progressivement.

Dans ce contexte, le choix de la méthodologie doit rester cohérent avec la nature évolutive du projet. Plusieurs approches de gestion de projet existent, allant des méthodes traditionnelles aux méthodes agiles. Une analyse comparative permet d'identifier celle qui répond le mieux aux exigences d'un système de détection Wi-Fi distribué à développer et à valider expérimentalement.

1.7.1 Les méthodes agiles

Les méthodes agiles reposent sur un développement itératif et incrémental. Le projet progresse par cycles courts, permettant une validation régulière des fonctionnalités et

une adaptation continue des choix techniques. Cette organisation favorise l'amélioration progressive du prototype et réduit les risques liés à une intégration tardive des différents composants.

Parmi les cadres agiles les plus utilisés figurent **Extreme Programming (XP)**, **Kanban** et **Scrum**.

Extreme Programming (XP)

XP met l'accent sur la qualité du code, les tests fréquents et l'amélioration continue des fonctionnalités. Cette approche est pertinente pour un développement logiciel intensif, mais elle structure moins clairement la planification globale d'un projet hybride combinant matériel, communication réseau et supervision.

Kanban

Kanban repose sur une gestion visuelle du flux de travail à l'aide d'un tableau organisé par états d'avancement. Elle offre une grande flexibilité et facilite le suivi des tâches. Toutefois, l'absence de cycles formels de validation peut limiter la structuration des livrables intermédiaires dans un projet nécessitant plusieurs phases d'expérimentation.

Scrum

Scrum est un cadre agile formalisé, organisé en cycles courts appelés *Sprints*, visant à produire des incréments fonctionnels successifs. Il repose sur les principes de transparence, d'inspection et d'adaptation. Ce cadre est particulièrement adapté lorsqu'un projet doit progresser par étapes validables, avec une visibilité claire sur les objectifs et les résultats obtenus.

TAB. 1.1 – Comparaison des principales méthodes agiles.

Méthode	Principe	Avantages	Limites
XP	Développement centré sur les tests et la qualité du code.	Amélioration continue du logiciel et forte discipline de développement.	Moins adaptée à la planification globale d'un projet hybride matériel/logiciel.
Kanban	Gestion visuelle et continue des tâches selon leur état d'avancement.	Grande flexibilité et bonne visibilité du flux de travail.	Absence de cycles formels de validation intermédiaire.
Scrum	Organisation du travail en Sprints planifiés avec objectifs définis.	Suivi structuré, validation progressive et meilleure maîtrise des étapes.	Nécessite une définition claire des objectifs et des priorités.

1.7.2 Choix de la méthode Scrum

Après analyse comparative, la méthode **Scrum** a été retenue pour la réalisation du projet Sentinel Hive. Le système développé combine des nœuds Raspberry Pi, une chaîne de communication MQTT, une journalisation centralisée et un moteur de corrélation Wazuh. Les performances du prototype dépendent d'essais expérimentaux successifs et certains paramètres peuvent être ajustés au cours du développement.

L'organisation en Sprints permet de valider chaque composant avant son intégration complète. Cette approche offre une meilleure maîtrise du planning, facilite la gestion des risques techniques et permet de suivre plus efficacement l'avancement du projet. Elle est donc adaptée à un prototype progressif dont la stabilité repose sur plusieurs phases de configuration, de test et de validation.

1.7.3 Présentation de Scrum

Conformément au cadre Scrum, l'organisation du projet peut être décrite à travers trois éléments fondamentaux : les rôles, les artefacts et l'organisation des Sprints.

Les rôles

Product Owner. Ce rôle correspond à l'entité qui porte la vision globale du projet, fixe les priorités et valide les résultats. Dans le cadre de Sentinel Hive, ce rôle est principalement assuré par la société d'accueil, qui oriente les besoins fonctionnels et apprécie la pertinence des choix réalisés.

Scrum Master. Le Scrum Master veille à la bonne application du cadre méthodologique, facilite l'avancement du travail et aide à lever les blocages. Dans ce projet, ce rôle est assuré dans un cadre académique simplifié à travers le suivi de l'encadrement et l'organisation structurée des étapes de réalisation.

Équipe de développement. L'implémentation technique est assurée par l'étudiant. Elle couvre la configuration des Raspberry Pi, la mise en place de la capture Wi-Fi, le développement des scripts Python/Scapy, l'intégration de la communication MQTT et la configuration de la supervision Wazuh.

Les artefacts

Product Backlog. Il regroupe l'ensemble des fonctionnalités et besoins nécessaires à l'évolution du prototype, comme la capture multi-canal, l'agrégation JSON, la journalisation et la génération d'alertes.

Sprint Backlog. Il rassemble les tâches retenues pour un Sprint donné ainsi que le plan de travail associé, par exemple la configuration des interfaces Wi-Fi, la validation du mode moniteur ou l'intégration du pipeline MQTT/Wazuh.

Incrément. Il correspond à la version fonctionnelle produite à la fin d'un Sprint. Dans ce projet, chaque incrément doit être exploitable, vérifiable et utile pour préparer l'étape suivante.

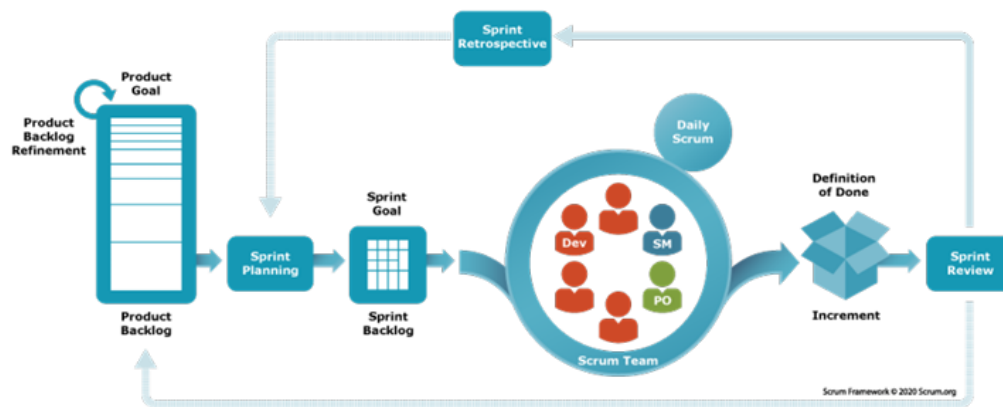


FIG. 1.2 – Cycle général du framework Scrum.

Organisation des Sprints

Le projet a été structuré en plusieurs étapes progressives :

- **Sprint 1.** Étude de l'existant, analyse du besoin et cadrage du projet ;
- **Sprint 2.** Mise en place de l'architecture de capture et structuration du prototype ;
- **Sprint 3.** Déploiement des nœuds, intégration de la communication et centralisation des données ;
- **Sprint 4.** Intégration du SIEM Wazuh, tests d'intrusion, mesures de performance et validation finale.

Cette organisation progressive a permis de développer le prototype de manière structurée tout en assurant une validation continue des performances et des choix techniques.

La planification relative du projet est synthétisée dans la figure suivante, qui présente une répartition des tâches par sprints et par grandes phases de réalisation.

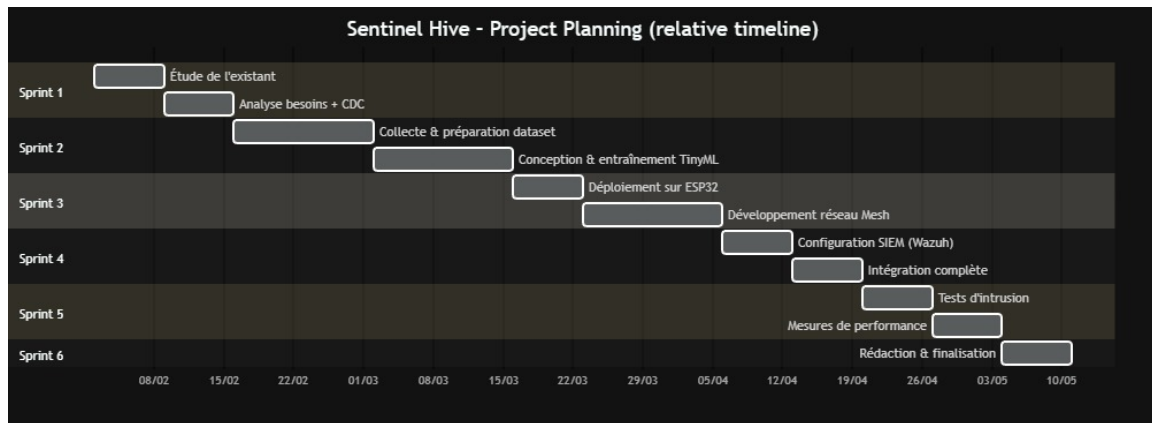


FIG. 1.3 – Planification relative du projet Sentinel Hive par sprints.

Les principales technologies mobilisées dans ce cadre sont les suivantes :

- **Matériel.** Raspberry Pi 4 et adaptateur Wi-Fi USB TP-Link AC600 ;
- **Capture et traitement.** Python, Scapy, mode moniteur et mode promiscuité ;
- **Communication.** MQTT pour le transport des messages ;
- **Supervision.** Mosquitto, agent Wazuh, Wazuh SIEM et dashboard ;
- **Environnement.** Linux Ubuntu 24.04, JSON et outils de journalisation.

Conclusion du chapitre

Ce chapitre a présenté le cadre général de Sentinel Hive en s'appuyant sur une structuration proche d'une véritable étude de projet. Le contexte du travail, l'organisme d'accueil, les problématiques techniques, les objectifs visés, la solution retenue et la méthodologie adoptée ont été détaillés de manière progressive. Cette base permet d'aborder, dans le chapitre suivant, l'étude du domaine et les fondements techniques avec une vision claire du besoin, des contraintes et de l'orientation générale de la solution.

Chapitre 2

État de l’art

2.1 Introduction

Le cadre général du projet étant posé, ce chapitre plonge dans les fondements techniques qui soutiennent **Sentinel Hive**. On ne va pas simplement lister des technologies — l’objectif est de comprendre pourquoi certaines approches tiennent la route en détection Wi-Fi et pourquoi d’autres tombent à plat dès qu’on sort du laboratoire. Les références retenues — Snort[2], Scapy[3], Wazuh[5] — ne sont pas là par hasard.

Ce qui ressort de l’analyse, c’est qu’une chaîne construite autour de Raspberry Pi[7], Scapy[3], MQTT[6], Mosquitto[4] et Wazuh[5] tient mieux ses promesses qu’une solution monolithique. Les solutions existantes ont des angles morts. Sentinel Hive est conçu précisément pour les couvrir.

2.2 Définition d’un système de détection Wi-Fi distribué

Un système de détection Wi-Fi distribué, c’est avant tout une architecture d’observation passive. Plusieurs points de capture, plusieurs canaux, un œil permanent sur les trames 802.11[9] qui circulent dans l’espace radio — sans jamais s’associer à un point d’accès. Ce n’est pas de la supervision IP classique. On descend au niveau de la couche liaison, là où des outils comme Kismet[10] ont montré qu’il se passe des choses intéressantes.

Le principe est simple en théorie : chaque nœud collecte localement, un système central corrèle et alerte. En pratique, c'est là que les architectures commencent à diverger. La couverture multizones, la résilience à la perte d'un capteur, la détection des attaques discrètes — désauthentification, disassociation, usurpation de point d'accès[11] — tout cela dépend de la qualité de l'organisation distribuée. Le référentiel MITRE ATT&CK[8] documente précisément ces modes opératoires.

2.3 Techniques de détection d'attaques Wi-Fi

Il n'existe pas une seule méthode de détection. Plusieurs approches coexistent, chacune avec ses angles d'attaque, et les architectures sérieuses en combinent au moins deux ou trois.

2.3.1 Détection par signatures

L'idée de base : on sait à quoi ressemble une attaque, on cherche ce motif dans le flux. Un volume anormal de trames de désauthentification, des rafales de requêtes de sondage, des structures de trames spécifiques — tout ça a une empreinte reconnaissable. La logique est directement héritée de ce que fait Snort[2] sur le trafic IP, appliquée aux scénarios référencés dans MITRE ATT&CK[8].

L'avantage ? La précision sur des attaques bien documentées est difficile à battre. La limite ? Dès qu'on sort du catalogue, ça ne détecte rien.

2.3.2 Détection par règles et corrélation

C'est ici que l'intégration dans un SIEM prend tout son sens. On ne cherche plus un motif fixe — on combine des compteurs, des seuils, des fréquences d'occurrence pour produire une alerte. Wazuh[5] est taillé pour ça. Un décodeur bien écrit transforme des statistiques brutes de trames en événements de sécurité traçables et explicables.

2.3.3 Détection par analyse comportementale

Plus difficile à calibrer, mais utile pour les anomalies diffuses. Variation soudaine du volume de trames, changement dans la nature du trafic, comportement incohérent d'un point d'accès — ces signaux faibles échappent aux approches par signature. Le vrai problème est qu'il faut une baseline solide du comportement normal de l'environnement[11]. Sans ça, les faux positifs s'accumulent.

2.3.4 Capture passive en mode moniteur

Techniquement, c'est le fondement de tout le reste. Une interface Wi-Fi en mode moniteur voit toutes les trames 802.11 [9] qui passent sur le canal surveillé — sans s'associer, sans injecter. Scapy[3] rend cette capture programmable et fine. Des événements invisibles depuis une collecte réseau classique deviennent soudainement observables.

2.3.5 Agrégation temporelle et interprétation

Transmettre chaque trame brute vers un système central, c'est ingérable à l'échelle. La solution? Agréger localement sur des fenêtres courtes — typiquement à la seconde — pour produire des statistiques légères. Le nœud de capture ne remonte pas un flux, il remonte une mesure. Ça change radicalement la charge sur le pipeline MQTT[6]/Mosquitto[4]/Wazuh[5].

2.4 Technologies de détection et de supervision utilisées

Le choix des briques technologiques n'est pas anodin. Légèreté, intégration, coût d'accès — ces trois critères ont guidé l'ensemble des décisions.

2.4.1 Raspberry Pi

Petit, peu gourmand, flexible. Le Raspberry Pi[7] n'est pas le hardware le plus puissant du marché, mais pour un capteur passif embarqué dans un prototype académique, il offre exactement ce qu'on cherche : un bon ratio coût/capacité, une communauté active et une intégration logicielle sans friction.

2.4.2 Scapy

Scapy[3] fait une chose et la fait bien : capturer, parser et manipuler des paquets réseau, jusqu'aux trames 802.11 en mode moniteur. C'est une bibliothèque Python, donc intégrable en quelques lignes. Elle laisse une liberté totale sur la logique d'agrégation — ce qui est exactement ce dont on a besoin pour un capteur personnalisé.

2.4.3 MQTT

MQTT[6] est pensé pour les environnements contraints. Faible overhead, transmission asynchrone, modèle publish/subscribe — c'est le protocole naturel pour faire remonter des mesures depuis des nœuds de terrain vers une infrastructure centrale sans saturer le réseau.

2.4.4 Mosquitto

Le broker de la chaîne. Mosquitto[4] reçoit, route, journalise. Sans lui, les nœuds Sentinel et Wazuh ne se parlent pas. C'est une brique intermédiaire discrète, mais retirer Mosquitto de l'architecture, c'est tout faire tomber.

2.4.5 Wazuh

Wazuh[5] assure la partie intelligente : décodeurs, règles de corrélation, visualisation. Il transforme des compteurs de trames Wi-Fi en alertes de sécurité interprétables. Dans

une logique SIEM, c'est lui qui fait la jonction entre les données brutes des capteurs et une analyse actionnable.

2.5 Domaines d'application des systèmes de détection

Wi-Fi

Les cas d'usage sont plus larges qu'on ne le pense au premier abord.

2.5.1 Maison connectée

Dans un environnement domestique intelligent, la continuité radio est critique. Une tentative de désauthentification[11] sur un thermostat ou un verrou connecté peut passer complètement inaperçue avec les outils grand public. Les techniques documentées dans MITRE ATT&CK[8] s'appliquent tout autant à ces environnements.

2.5.2 PME et ateliers connectés

Les petites structures n'ont ni les moyens ni le personnel pour déployer une infrastructure SOC complète. Une architecture légère comme Sentinel Hive — Raspberry Pi[7], Scapy[3], coût maîtrisé — leur donne une visibilité précoce sur leur environnement radio sans investissement disproportionné.

2.5.3 Laboratoires pédagogiques

C'est un terrain d'expérimentation idéal. Capture de trames avec Scapy[3], transport MQTT[6], supervision Wazuh[5] sur Raspberry Pi[7] — chaque brique illustre un concept concret de cybersécurité Wi-Fi. L'ensemble forme un support de TP complet.

2.5.4 Événementiel et infrastructures temporaires

Un salon professionnel, une démonstration technique, une installation ponctuelle — ces environnements ont besoin d'une solution rapidement déployable, autonome et reconfigurable. Raspberry Pi[7] et MQTT[6] répondent naturellement à ce besoin.

2.5.5 Analyse et justification du choix technologique

Le besoin est clair : légèreté, reproductibilité, observation directe des trames Wi-Fi. L'architecture Raspberry Pi / Scapy / MQTT / Mosquitto / Wazuh répond point par point — modulaire, peu coûteuse, et capable de couvrir à la fois la collecte locale et la supervision centralisée sans dépendance à un vendor propriétaire.

2.6 Architecture générale de la solution proposée

L'architecture de Sentinel Hive se décompose en couches fonctionnelles distinctes, chacune avec une responsabilité claire.

2.6.1 Vue d'ensemble

L'architecture de Sentinel Hive s'appuie sur des nœuds Raspberry Pi[7] dédiés à la capture passive des trames Wi-Fi. Les événements collectés sont transmis vers l'infrastructure centrale au moyen du protocole MQTT[6] et du broker Mosquitto, puis exploités par Wazuh[5] pour la corrélation et la visualisation des alertes de sécurité. En complément, une machine virtuelle Ubuntu héberge une plateforme web légère de monitoring, composée d'un backend Flask, d'une collecte de métriques avec *psutil*, d'une API REST exposant les données au format JSON et d'un tableau de bord Chart.js pour l'affichage dynamique des ressources système. Cette séparation des responsabilités rend le prototype extensible et permet de faire évoluer chaque couche indépendamment.

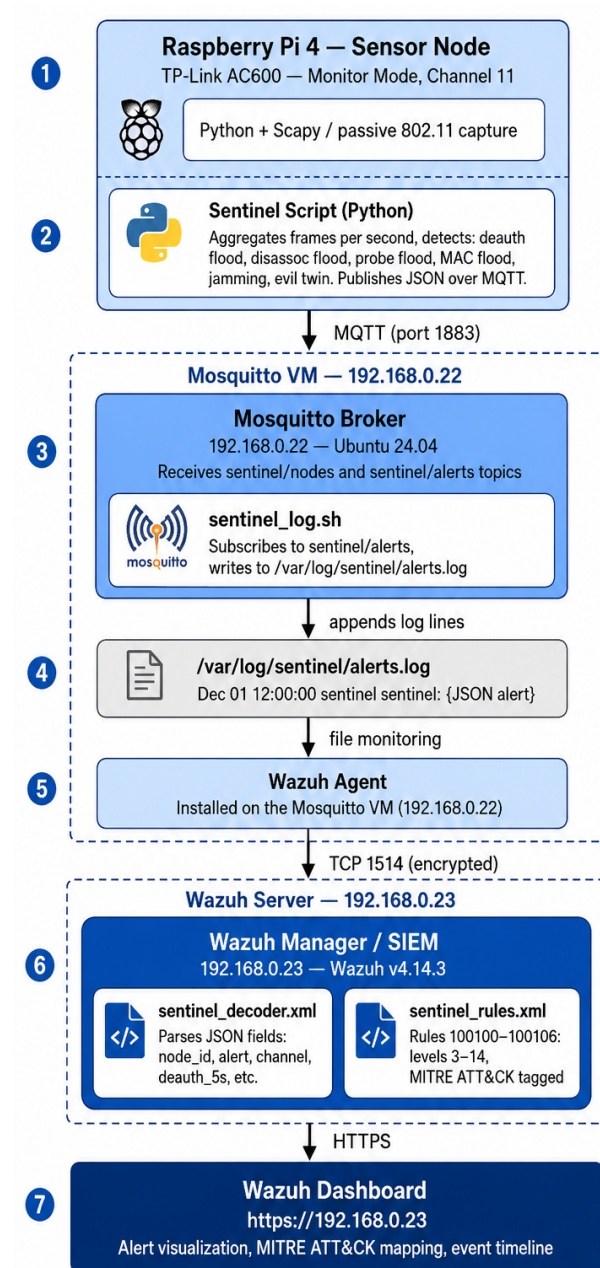


FIG. 2.1 – Architecture générale de la solution Sentinel Hive.

2.6.2 Couche acquisition : nœuds de capture

Des Raspberry Pi configurés en mode moniteur, chacun affecté à un canal 802.11 [9] spécifique. L'idée est de couvrir une portion ciblée du spectre radio sans se marcher dessus entre nœuds.

2.6.3 Couche traitement local : agrégation des trames

Scripts Python, Scapy[3], fenêtres temporelles courtes. Le nœud ne fait pas de l'intelligence — il compte, agrège, produit des statistiques. Le traitement lourd est délégué à l'aval.

2.6.4 Couche communication : transport des mesures

Des messages JSON via MQTT[6]. Léger, fiable, asynchrone. Les nœuds de terrain n'ont pas besoin de savoir ce qui se passe côté supervision — ils publient, c'est tout.

2.6.5 Couche journalisation : broker Mosquitto

Mosquitto[4] concentre les flux, assure la traçabilité. Chaque message qui remonte des nœuds passe par lui. C'est le point d'entrée de l'infrastructure centrale.

2.6.6 Couche supervision : décodage et corrélation

Wazuh[5] applique ses décodeurs et règles. Des compteurs de trames deviennent des alertes de sécurité — c'est la transformation clé de l'architecture.

2.6.7 Couche visualisation : tableau de bord et analyse

Le dashboard Wazuh[5] offre la vue centralisée. Consultation des événements, analyse des alertes, interprétation de l'activité radio — tout en un seul endroit.

2.6.8 Couche monitoring système : plateforme web dédiée

Afin de compléter la supervision de sécurité assurée par Wazuh, Sentinel Hive intègre également une interface web personnalisée dédiée au suivi des ressources de l'infrastructure centrale. Cette couche repose sur une machine virtuelle Ubuntu Server, un backend Flask développé en Python, la bibliothèque *psutil* pour la collecte des métriques système,

ainsi qu'une interface HTML, CSS et JavaScript exploitant Chart.js pour la visualisation graphique.

Le flux de fonctionnement de cette couche peut être résumé comme suit : **Ubuntu VM** → **Flask backend** → **collecte psutil** → **API REST** → **tableau de bord Chart.js**. Cette organisation permet d'observer en temps réel la consommation CPU, l'utilisation de la mémoire RAM et l'activité réseau de la machine virtuelle qui supporte les services centraux du prototype. La plateforme ne remplace pas Wazuh ; elle apporte une visibilité complémentaire sur l'état opérationnel de l'environnement de déploiement.

2.7 Étude de marché des solutions existantes

Trois solutions représentatives permettent de situer Sentinel Hive dans son environnement concurrentiel.

2.7.1 Snort : solution open source de type NIDS

Description. Snort[1, 2] est le NIDS open source de référence. Moteur de règles puissant, large communauté, documentation abondante.

Avantages

- solution mature, bien rodée en environnement SOC ;
- personnalisation des règles très poussée ;
- intégration SIEM classique sans friction.

Limites

- trop gourmand en ressources pour un microcontrôleur ou un Raspberry Pi ;
- logique signatures-first — aveugle face aux comportements inconnus ;
- pas pensé pour les contraintes radio IoT[2].



FIG. 2.2 – Plateforme Snort.

2.7.2 Fingbox : solution orientée grand public

Description. Fingbox[12] cible les environnements domestiques et TPE. Découverte d'équipements, alertes de base, interface mobile — c'est l'outil du non-expert.

Avantages

- déploiement immédiat, zéro friction pour l'utilisateur final ;
- bonne visibilité sur les appareils du réseau ;
- interface claire et accessible.

Limites

- dépendance cloud forte — pas de maîtrise des données ;
- quasi-inextensible pour un usage recherche ;
- face aux attaques sophistiquées, les capacités de détection sont anémiques.



FIG. 2.3 – Plateforme Fingbox.

2.7.3 Armis : plateforme entreprise agentless

Description. Armis[13] joue dans une autre cour — visibilité multi-actifs à grande échelle, détection comportementale, intégration SOC entreprise. Pas d'agent à déployer sur chaque terminal.

Avantages

- couverture d'actifs hétérogènes difficile à évaluer ;
- détection comportementale adaptée aux environnements complexes ;
- s'intègre naturellement dans un SOC d'entreprise existant.

Limites

- le ticket d'entrée est prohibitif pour un prototype académique ;
- infrastructure centralisée imposée — peu de flexibilité ;
- moins personnalisable qu'une chaîne ouverte Raspberry Pi / Python.



FIG. 2.4 – Plateforme Armis.

2.7.4 Comparaison synthétique et justification

TAB. 2.1 – Comparaison synthétique des plateformes existantes.

Critère	Snort	Fingbox	Armis	Justification pour Sentinel Hive
Capture 802.11 brute	Moyenne	Faible	Moyenne	Sentinel Hive privilégie la capture passive native des trames Wi-Fi via Scapy en mode moniteur.
Ouverture de l'architecture	Bonne	Faible	Moyenne	L'utilisation de Raspberry Pi, MQTT et Wazuh facilite l'audit, l'extension et la reproductibilité.
Coût d'accès	Faible	Moyen	Élevé	Une architecture open source limite les coûts de déploiement.
Détection Wi-Fi spécifique	Moyenne	Faible	Bonne	Les règles Wazuh[5] peuvent être adaptées à des signatures de floods 802.11 et d'Evil Twin décrites dans MITRE ATT&CK[8].
Intégration SIEM	Bonne	Limitée	Bonne	Wazuh centralise les logs Mosquitto et applique les décodeurs/règles personnalisés.

2.8 Conclusion

Ce chapitre a tracé le périmètre technique de Sentinel Hive : les techniques de détection Wi-Fi qui tiennent réellement la route, les technologies retenues et pourquoi elles font sens ensemble, les domaines d'application concrets et les limites des solutions existantes. Snort est trop lourd, Fingbox trop fermé, Armitis trop cher. La chaîne Scapy[3] / MQTT[6] / Mosquitto[4] / Wazuh[5] n'est pas un compromis par défaut — c'est un choix argumenté, fondé sur les contraintes réelles du projet.

Chapitre 3

Spécification des besoins de sécurité et plan de validation

3.1 Introduction

Dans un système de détection d'intrusion Wi-Fi distribué, la sécurité ne concerne pas uniquement les attaques observées sur le réseau sans fil. Elle concerne également la fiabilité des capteurs, l'intégrité des données remontées, la disponibilité de la chaîne de supervision et la confidentialité des informations collectées. Pour le projet **Sentinel Hive**, il est donc nécessaire de définir clairement les besoins de sécurité, les exigences associées, l'architecture de protection retenue ainsi que les mécanismes de validation du prototype.

Ce chapitre présente d'abord les besoins de sécurité du système, puis formalise les exigences fonctionnelles et non fonctionnelles qui en découlent. Il décrit ensuite l'architecture de sécurité proposée, la politique de sécurité applicable au prototype ainsi qu'un plan de tests destiné à vérifier le bon fonctionnement et la robustesse de la solution.

3.2 Outils et technologies utilisés

La réalisation de Sentinel Hive s'appuie sur un ensemble d'outils couvrant la conception, le développement, la capture Wi-Fi, la communication, la supervision et la validation

expérimentale. Le tableau suivant synthétise les technologies retenues ainsi que leur rôle dans le prototype.

TAB. 3.1 – Outils et technologies utilisés dans Sentinel Hive – partie 1.

Logo	Outil / technologie	Description et rôle dans le projet
	Wazuh	Plateforme SIEM utilisée pour le décodage, la corrélation et la visualisation des alertes produites à partir des événements Sentinel Hive.
	Visual Studio Code	Environnement de développement utilisé pour éditer les scripts Python, les fichiers de configuration et les composants web de la plateforme.
	StarUML	Outil de modélisation utilisé pour réaliser les diagrammes UML du système, notamment les cas d'utilisation et les interactions entre composants.
	Mosquitto	Broker MQTT chargé de recevoir, centraliser et journaliser les événements publiés par les nœuds de capture Sentinel Hive.
	MQTT	Protocole léger et asynchrone utilisé pour transporter les messages JSON entre les capteurs distribués et l'infrastructure centrale.
	Scapy	Bibliothèque Python utilisée pour la capture passive, l'analyse des trames 802.11 et l'extraction des indicateurs radio nécessaires à la détection.
	Python	Langage principal du prototype, utilisé pour les scripts de capture, le prétraitement local, la publication des événements et le backend Flask.
	Flask	Framework web utilisé pour développer le backend de la plateforme de monitoring et exposer les métriques système via une API REST.

TAB. 3.2 – Outils et technologies utilisés dans Sentinel Hive – partie 2.

Logo	Outil / technologie	Description et rôle dans le projet
	psutil	Bibliothèque Python utilisée pour collecter les métriques CPU, RAM et réseau de la machine virtuelle Ubuntu en temps réel.
	HTML	Langage de structuration utilisé pour construire l'interface web du tableau de bord de monitoring personnalisé.
	CSS	Langage de style utilisé pour organiser l'affichage, améliorer la lisibilité et homogénéiser l'interface graphique du dashboard.
	JavaScript	Langage utilisé côté frontend pour interroger l'API Flask de manière asynchrone et mettre à jour dynamiquement les indicateurs.
	Chart.js	Bibliothèque JavaScript utilisée pour afficher les graphiques temps réel liés au CPU, à la RAM et à l'activité réseau.
	Ubuntu Server	Système d'exploitation utilisé pour héberger les services centraux du prototype : Mosquitto, Wazuh et la plateforme web de monitoring.
	Kali Linux	Environnement de test utilisé dans un laboratoire contrôlé pour générer les scénarios d'attaque Wi-Fi et valider la détection.

3.3 Analyse des besoins de sécurité

3.3.1 Protection de l'infrastructure de capture

Les nœuds Raspberry Pi déployés sur le terrain constituent des composants critiques de l'architecture. Ils doivent être protégés contre l'altération locale de leur configuration, l'arrêt non autorisé des services, l'exécution de scripts malveillants et la modification des paramètres de capture. Un besoin essentiel consiste donc à garantir l'intégrité logicielle des capteurs et la stabilité de leur fonctionnement.

3.3.2 Intégrité des événements remontés

Les messages produits par les capteurs servent de base à l'analyse et à la détection via MQTT et Wazuh[6, 5]. Toute modification, injection ou suppression de ces événements pendant leur transit compromet la qualité de la supervision. Le système doit ainsi garantir que les données publiées via MQTT restent cohérentes, correctement structurées et traçables jusqu'à leur réception par la plateforme centrale.

3.3.3 Disponibilité de la chaîne de supervision

La valeur du prototype repose sur sa capacité à observer le milieu radio en continu et à remonter rapidement les alertes[9, 5]. Le système doit donc limiter les interruptions de service, supporter des montées ponctuelles de charge et continuer à fonctionner même en cas de perte temporaire d'un nœud de capture. La disponibilité concerne à la fois les capteurs, le broker MQTT, la journalisation et le moteur Wazuh.

3.3.4 Confidentialité et maîtrise des accès

Même si Sentinel Hive ne vise pas à intercepter le contenu déchiffré des communications, les métadonnées collectées peuvent révéler des informations sensibles sur l'activité radio, les identifiants d'équipements ou la chronologie des événements. Il est donc néces-

saire de restreindre l'accès aux données, aux tableaux de bord et aux fichiers journaux aux seuls utilisateurs autorisés.

3.3.5 Traçabilité et auditabilité

Une plateforme de supervision doit permettre de comprendre l'origine d'une alerte, de retracer le cheminement d'un événement et d'identifier les actions d'administration effectuées sur le système. Le projet nécessite par conséquent une traçabilité minimale couvrant la publication des messages, leur journalisation, leur décodage et leur corrélation au sein de Wazuh.

3.4 Exigences de sécurité du système

3.4.1 Exigences fonctionnelles

À partir des besoins précédents, plusieurs exigences fonctionnelles peuvent être définies pour le prototype.

- le système doit capturer passivement les trames Wi-Fi et produire des événements agrégés exploitables ;
- chaque nœud Sentinel doit transmettre ses observations vers le broker MQTT à travers un format structuré et uniforme ;
- la plateforme centrale doit journaliser les événements reçus et permettre leur exploitation par Wazuh ;
- des règles de détection doivent permettre de générer des alertes en présence de comportements anormaux, comme un nombre élevé de trames de désauthentification ou l'apparition d'un point d'accès suspect ;
- le système doit permettre l'identification du nœud émetteur, de l'horodatage et du type d'événement observé ;
- les interfaces d'administration doivent être accessibles uniquement après authentification.

3.4.2 Exigences non fonctionnelles

Les exigences non fonctionnelles complètent les exigences précédentes en encadrant la qualité globale de la solution.

- **Disponibilité.** Les services essentiels doivent démarrer automatiquement et reprendre après redémarrage des équipements ;
- **Intégrité.** Les messages échangés doivent conserver leur structure attendue et être rejetés s'ils sont invalides ou incohérents ;
- **Performance.** La chaîne doit supporter une remontée régulière d'événements sans saturation excessive du broker ni perte majeure d'information ;
- **Évolutivité.** L'ajout d'un nouveau nœud de capture doit rester simple et ne pas nécessiter de refonte de l'architecture ;
- **Maintenabilité.** Les composants doivent être configurables, documentés et facilement supervisables ;
- **Auditabilité.** Toute alerte importante doit pouvoir être reliée à une source, à un horodatage et à une règle de détection.

3.4.3 Hiérarchisation des exigences

Toutes les exigences n'ont pas le même niveau de criticité. Dans le cadre de Sentinel Hive, la disponibilité de la collecte, l'intégrité des événements et la capacité de détection constituent les priorités principales. Viennent ensuite la traçabilité, la maîtrise des accès et la facilité d'extension du système. Cette hiérarchisation permet d'orienter les choix techniques du prototype vers les fonctions les plus déterminantes pour la démonstration de faisabilité.

3.5 Architecture de sécurité proposée

3.5.1 Vue d'ensemble

L'architecture de sécurité du projet s'appuie sur une séparation claire entre les fonctions de capture, de transport, de centralisation et d'analyse[6, 4, 5]. Les nœuds Raspberry Pi[7] assurent uniquement l'observation radio et la génération d'événements. Le broker MQTT[6] reçoit et relaie les messages. Les logs produits sont ensuite exploités par Wazuh[5] pour le décodage, la corrélation et la génération des alertes. Cette séparation réduit la complexité de chaque composant et facilite l'analyse des flux de confiance.

3.5.2 Sécurisation des nœuds Sentinel

Chaque nœud doit être configuré avec un système minimal, des services limités au strict nécessaire et des mécanismes d'accès restreints. Les interfaces inutiles doivent être désactivées, les comptes par défaut remplacés, les mots de passe renforcés et l'accès distant protégé. Les scripts de capture doivent être exécutés avec des privilèges maîtrisés et supervisés afin de réduire les risques liés à une compromission locale.

3.5.3 Sécurisation de la communication MQTT

La communication entre les capteurs et la plateforme centrale doit être encadrée par une politique de publication claire. Les topics utilisés doivent être organisés par nœud et par type d'événement afin de faciliter le contrôle des flux. L'authentification des clients MQTT, la restriction des droits de publication et, lorsque cela est possible, l'utilisation de TLS permettent de réduire les risques d'usurpation, d'écoute ou d'injection de messages.

3.5.4 Journalisation centralisée et corrélation

Mosquitto et les composants intermédiaires doivent produire des journaux exploitables afin de conserver une trace des connexions, publications et anomalies de protocole. Ces

journaux alimentent Wazuh, qui applique des décodeurs adaptés au format JSON retenu et déclenche des règles corrélant les événements issus des différents nœuds. Cette organisation améliore la visibilité opérationnelle et permet une analyse unifiée des signaux radio observés.

3.5.5 Segmentation logique de l'architecture

Même dans un prototype académique, une séparation logique des rôles améliore la sécurité globale. Le réseau de gestion, le canal de transport des événements et la console de supervision doivent être distingués autant que possible. Cette segmentation limite l'impact d'un incident local sur l'ensemble de la chaîne et facilite l'application de contrôles d'accès spécifiques selon les composants.

3.6 Diagrammes UML du système

Cette section regroupe les principaux diagrammes UML utilisés pour décrire le fonctionnement de Sentinel Hive. Afin de garder une logique Scrum claire, les diagrammes sont présentés par Sprint, avec le backlog associé à chaque étape.

3.6.1 Sprint 1 : analyse fonctionnelle et cadrage

Le premier Sprint porte sur la compréhension du besoin, l'identification des acteurs et la description des interactions principales du système. Il sert à clarifier le périmètre fonctionnel avant de passer à la conception détaillée.

TAB. 3.3 – Sprint Backlog détaillé du Sprint 1.

ID	User story / tâche	Priorité	Travail réalisé	Livrable attendu
S1.1	Identifier les acteurs du système	Haute	Définir les rôles de l'administrateur, du capteur Sentinel, du broker MQTT et de la plateforme Wazuh.	Liste des acteurs et responsabilités clairement formalisées.
S1.2	Définir les cas d'utilisation principaux	Très haute	Décrire la capture Wi-Fi, la remontée des événements, la consultation des alertes et l'administration de la solution.	Diagramme de cas d'utilisation validant le périmètre fonctionnel.
S1.3	Prioriser les besoins fonctionnels	Haute	Distinguer les fonctions indispensables des fonctions secondaires afin de cadrer le prototype réalisable.	Besoins classés par priorité et reliés aux objectifs du projet.
S1.4	Décrire le scénario nominal de détection	Très haute	Formaliser l'enchaînement entre capteur, MQTT, logs, Wazuh Manager, Indexer et Dashboard.	Diagramme de séquence décrivant le flux de détection bout en bout.
S1.5	Valider le périmètre du Sprint	Moyenne	Vérifier que les cas d'utilisation et les échanges couvrent le besoin de supervision Wi-Fi distribuée.	Base fonctionnelle stabilisée avant la conception technique.

Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation présente les interactions principales entre l'administrateur, le système Sentinel Hive et les fonctions de supervision offertes par la plateforme.

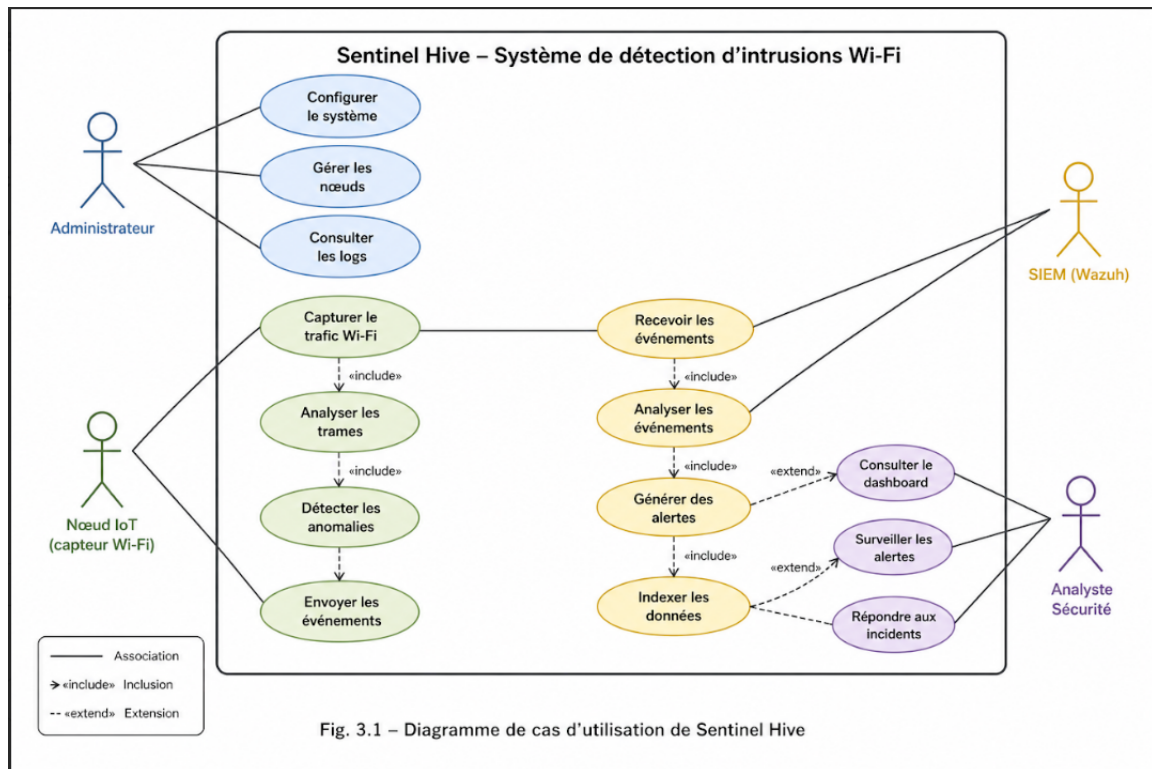


Fig. 3.1 – Diagramme de cas d'utilisation de Sentinel Hive

FIG. 3.1 – Diagramme de cas d'utilisation de Sentinel Hive.

Diagramme de séquence du processus de détection

Ce diagramme décrit le scénario principal de détection dans Sentinel Hive. Il met en évidence l'enchaînement entre le nœud capteur, le broker MQTT et la plateforme Wazuh, depuis la capture locale des événements Wi-Fi jusqu'à leur transmission vers l'environnement central.

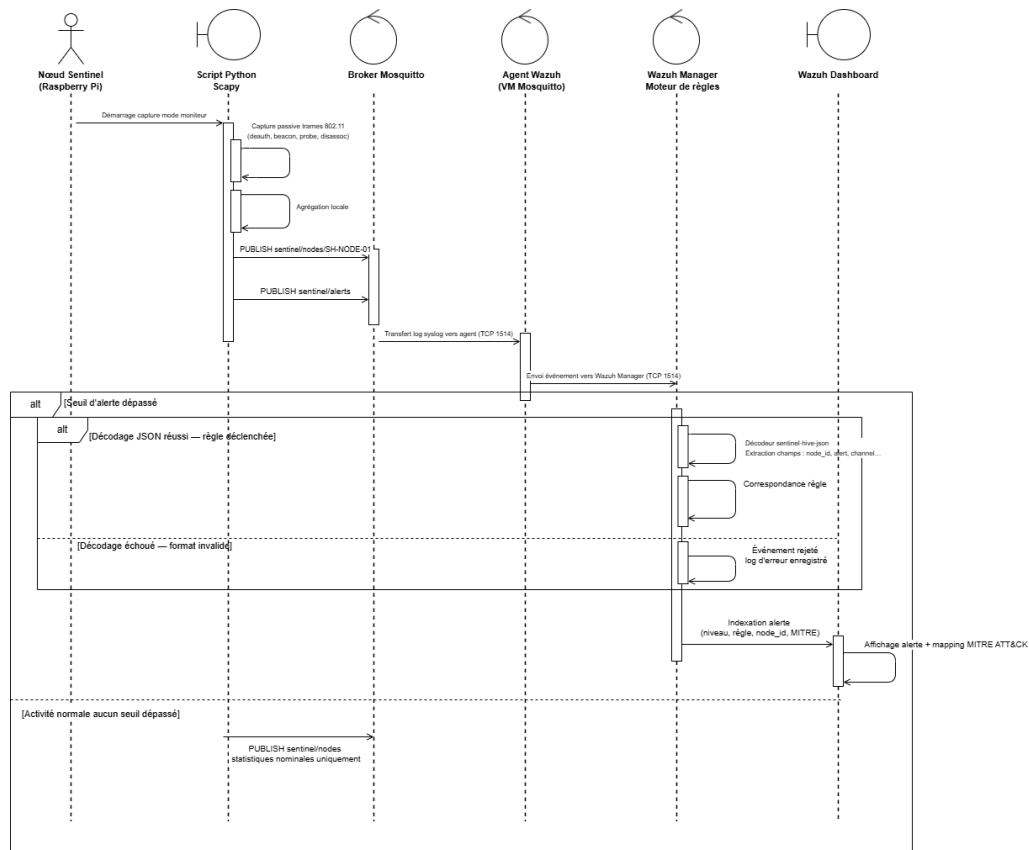


FIG. 3.2 – Diagramme de séquence du processus de détection.

Diagramme de séquence du processus d'authentification

Ce diagramme présente le déroulement de l'accès à la plateforme de supervision. Il illustre les échanges liés à la saisie des identifiants, à la vérification des informations d'authentification et à l'ouverture d'une session permettant à l'utilisateur d'accéder aux fonctionnalités de consultation.

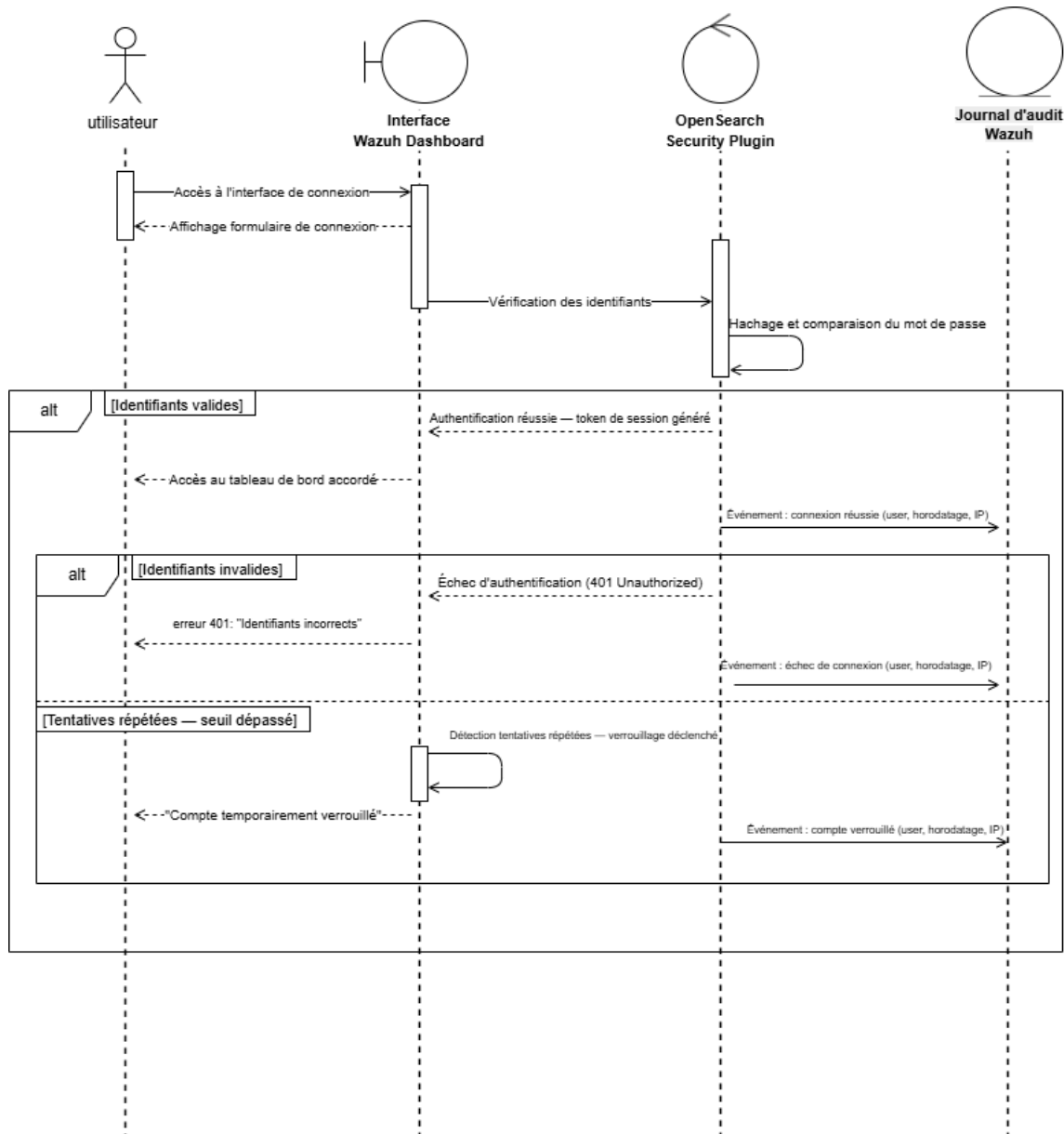


FIG. 3.3 – Diagramme de séquence du processus d'authentification et d'accès à la plateforme.

3.6.2 Sprint 2 : conception technique et structuration du prototype

Le deuxième Sprint transforme le cadrage fonctionnel en conception technique. Il précise les entités manipulées, leurs relations et le déroulement interne du traitement depuis la capture jusqu'à l'affichage des alertes.

TAB. 3.4 – Sprint Backlog détaillé du Sprint 2.

ID	User story / tâche	Priorité	Travail réalisé	Livrable attendu
S2.1	Structurer les entités logicielles	Très haute	Identifier les classes liées au capteur, au moteur de détection, aux seuils, aux événements, au broker et à Wazuh.	Diagramme de classes représentant la structure logique du prototype.
S2.2	Définir le modèle d'événement	Haute	Préciser les champs utiles : type d'anomalie, horodatage, canal, source, niveau et message transmis.	Format d'événement cohérent avec l'agrégation locale et l'ingestion Wazuh.
S2.3	Modéliser le traitement local	Très haute	Décrire la capture, l'analyse des trames, la classification de l'anomalie et la génération du log structuré.	Diagramme d'activité montrant le déroulement complet du traitement.
S2.4	Préparer l'intégration MQTT/Wazuh	Haute	Relier les sorties du capteur aux mécanismes de publication, journalisation, décodage et corrélation.	Chaîne technique prête pour l'implémentation et les tests d'intégration.
S2.5	Définir les critères de validation technique	Moyenne	Associer chaque composant à un résultat vérifiable : capture active, message reçu, log présent, alerte visible.	Critères de test exploitables pour la phase de validation expérimentale.

Diagramme de classes

Le diagramme de classes met en évidence la structuration logique des entités manipulées dans la solution ainsi que leurs relations au sein du prototype.

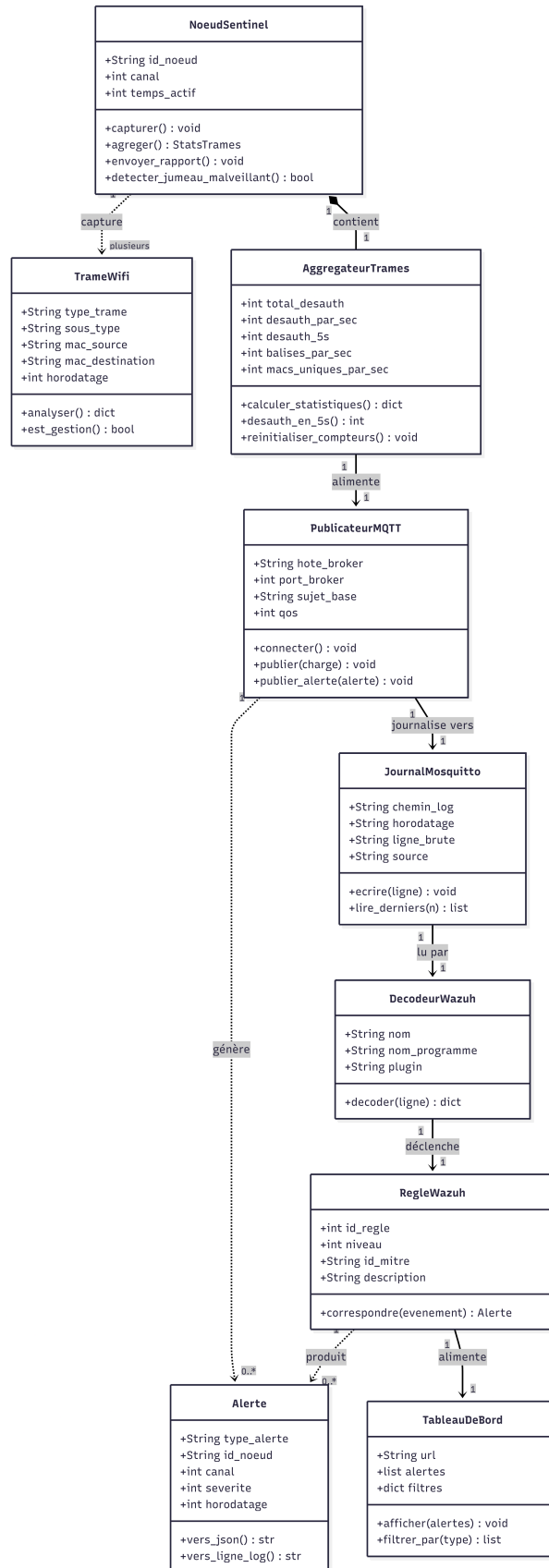


FIG. 3.4 – Diagramme de classes de la solution.

Diagramme d'activité

Le diagramme d'activité décrit le déroulement global du traitement, depuis la capture locale des trames jusqu'à la génération et la consultation des alertes dans l'interface de supervision.

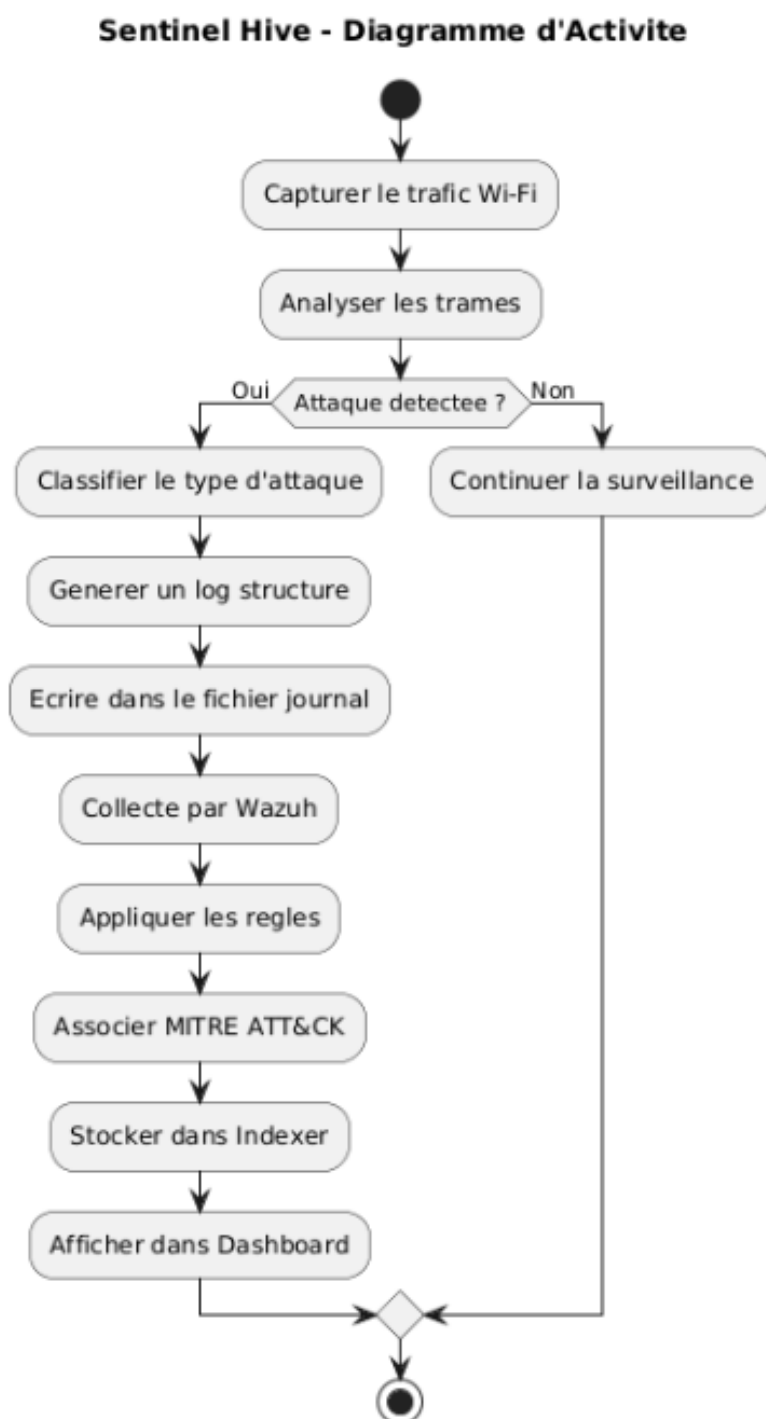


FIG. 3.5 – Diagramme d'activité du système Sentinel Hive.

3.7 Politique de sécurité du prototype

3.7.1 Gestion des accès

L'accès aux Raspberry Pi, au broker MQTT et à la plateforme Wazuh doit être réservé aux utilisateurs autorisés. Les comptes partagés doivent être évités, les identifiants par défaut supprimés et les mots de passe définis selon un niveau de complexité suffisant. Toute opération d'administration doit être réalisée via des comptes identifiés et, si possible, par des canaux sécurisés.

3.7.2 Durcissement des équipements

Les équipements déployés doivent faire l'objet d'un durcissement minimal : mise à jour régulière du système, désactivation des services non nécessaires, limitation des ports exposés et contrôle des fichiers de configuration. Les capteurs doivent démarrer avec une configuration stable et reproductible, afin de réduire les écarts entre les nœuds du prototype.

3.7.3 Gestion des journaux et de la conservation des données

Les journaux doivent être conservés de manière cohérente pour permettre l'analyse des incidents et la vérification du bon fonctionnement du système. Une durée de conservation adaptée au contexte du projet doit être définie, avec une rotation des fichiers afin d'éviter la saturation du stockage. Les données collectées doivent rester limitées à ce qui est nécessaire pour l'objectif de détection et d'expérimentation.

3.7.4 Réaction en cas d'incident

En cas d'anomalie détectée, la procédure minimale consiste à identifier le nœud concerné, vérifier la cohérence des événements remontés, consulter les journaux du broker et de Wazuh puis isoler, si nécessaire, le composant suspect. Dans le cadre

du prototype, cette politique vise surtout à assurer une reprise rapide du service et à conserver les éléments utiles à l'analyse de l'incident.

3.7.5 Bonnes pratiques d'exploitation

Le maintien du niveau de sécurité du prototype suppose également une discipline d'exploitation : sauvegarde des configurations, documentation des changements, contrôle des versions de scripts, vérification régulière des règles de détection et tests périodiques de remontée d'alertes. Ces pratiques contribuent à la fiabilité du système tout au long des expérimentations.

3.8 Plan de validation et de tests de sécurité

3.8.1 Objectifs du plan de test

Le plan de validation a pour objectif de vérifier que la solution répond aux besoins de sécurité identifiés, que les composants communiquent correctement entre eux et que les alertes attendues sont bien produites dans les scénarios ciblés. Il permet aussi de mesurer les limites du prototype et de préparer les améliorations futures.

3.8.2 Stratégie générale de validation

La validation doit être menée de manière progressive. Une première étape porte sur les tests unitaires de configuration de chaque composant : mode moniteur sur les capteurs, publication MQTT, réception côté broker et collecte des journaux. Une seconde étape vérifie l'intégration de bout en bout. Enfin, des scénarios de sécurité contrôlés permettent de s'assurer que les règles de détection réagissent correctement aux événements générés.

3.8.3 Jeux de tests proposés

Les principaux tests à réaliser peuvent être structurés comme suit. Ils couvrent les dimensions essentielles du prototype, depuis la configuration initiale des capteurs jusqu'à la validation de la corrélation dans Wazuh. Cette organisation permet d'évaluer séparément chaque maillon de la chaîne, puis de vérifier la cohérence globale du système dans des scénarios de sécurité représentatifs.

TAB. 3.5 – Plan synthétique de validation et de tests de sécurité.

Catégorie	Test	Objectif	Résultat attendu
Configuration	Validation du mode moniteur	Vérifier que l'interface Wi-Fi observe correctement les trames 802.11	Les trames de gestion sont capturées sans association à un point d'accès
Communication	Publication MQTT authentifiée	Vérifier la remontée correcte d'un message structuré par un nœud Sentinel	Le broker accepte un message valide et rejette un client non autorisé
Journalisation	Contrôle des logs Mosquitto	Vérifier la traçabilité des connexions et publications	Les connexions et topics publiés apparaissent dans les journaux
Corrélation	Décodage Wazuh d'un événement JSON	Vérifier l'interprétation correcte du message remonté	L'événement est parsé, enrichi et classé selon les règles définies
Détection	Simulation d'un flood de désauthentification	Vérifier la génération d'une alerte face à un volume anormal de trames	Une alerte de niveau approprié est générée dans Wazuh
Robustesse	Arrêt d'un nœud de capture	Observer la résilience du système en cas de perte d'un capteur	Le système continue à fonctionner avec les autres nœuds et l'incident est identifiable
Accès	Tentative de connexion non autorisée	Vérifier l'application des contrôles d'accès sur les services	L'accès est refusé et l'événement est journalisé

3.8.4 Critères d'acceptation

Le prototype sera considéré comme valide si les capteurs transmettent correctement leurs événements, si la plateforme centrale journalise et corrèle ces messages de manière exploitable, si les règles de détection remontent les alertes prévues dans les scénarios

retenus et si les mécanismes de contrôle d'accès empêchent les usages non autorisés les plus évidents. Les résultats devront en outre rester reproductibles sur plusieurs cycles de test.

3.8.5 Limites de la validation

La validation d'un prototype académique ne couvre pas l'ensemble des cas d'usage d'une plateforme industrielle. Les tests seront réalisés dans un environnement maîtrisé, avec un nombre limité de nœuds et des scénarios d'attaque ciblés. Les résultats devront donc être interprétés comme une preuve de faisabilité et non comme une garantie exhaustive de sécurité en environnement réel.

3.9 Conclusion

Ce chapitre a permis de formaliser les besoins de sécurité de Sentinel Hive, de définir les exigences principales du système, de proposer une architecture de protection cohérente et d'établir une politique de sécurité adaptée au prototype. Il a également présenté un plan de validation destiné à vérifier la fiabilité de la chaîne de collecte, de transport, de corrélation et d'alerte. Cette spécification servira de référence pour la phase de réalisation et pour l'évaluation expérimentale du système.

Chapitre 4

Environnements de déploiement et scénario d'exécution

4.1 Introduction

Ce chapitre présente les environnements de déploiement mobilisés par **Sentinel Hive** et décrit un scénario d'exécution représentatif mettant en évidence la chaîne complète de traitement du prototype. L'objectif est de montrer, dans un cadre expérimental maîtrisé, comment une anomalie radio Wi-Fi est captée localement, prétraitée sur le nœud de surveillance, transmise vers l'infrastructure centrale, corrélée par le SIEM puis restituée sous forme d'indicateurs et d'alertes exploitables.

Le chapitre adopte une logique de démonstration technique orientée **attaque** → **détection** → **corrélation** → **visualisation**. Il vise ainsi à valider non seulement l'intégration fonctionnelle des composants, mais aussi la pertinence de l'approche distribuée retenue pour Sentinel Hive dans le contexte de la détection d'intrusions Wi-Fi.

4.2 Sprint 3 : déploiement de l'environnement technique

Ce troisième Sprint est consacré à la préparation et au déploiement des environnements nécessaires au fonctionnement de Sentinel Hive. Il couvre la mise en place du capteur, du

broker MQTT, de la journalisation Mosquitto et de la plateforme Wazuh afin de disposer d'une chaîne technique opérationnelle avant l'exécution des scénarios de validation.

TAB. 4.1 – Sprint Backlog détaillé du Sprint 3.

ID	User story / tâche	Priorité	Travail réalisé	Livrable attendu
S3.1	Préparer le nœud capteur	Très haute	Configurer l'environnement Linux, l'interface Wi-Fi en mode moniteur et les dépendances nécessaires à la capture.	Capteur prêt à observer les trames 802.11 sur le canal défini.
S3.2	Déployer le module de capture	Très haute	Exécuter le script Python/Scapy chargé de filtrer, agréger et structurer les événements radio.	Événements locaux générés sous un format exploitable.
S3.3	Mettre en place le broker MQTT	Haute	Installer et configurer Mosquitto, les topics de publication et les contrôles d'accès élémentaires.	Broker opérationnel recevant les messages des capteurs.
S3.4	Préparer l'intégration Wazuh	Haute	Configurer la journalisation, les décodeurs et les règles nécessaires à l'ingestion des événements Sentinel Hive.	Plateforme Wazuh prête à corréler les événements remontés.
S3.5	Vérifier la chaîne technique	Moyenne	Contrôler la continuité entre capture locale, publication MQTT, logs Mosquitto et ingestion SIEM.	Environnement de déploiement validé avant les essais d'attaque.

4.2.1 Présentation des environnements techniques

4.2.2 Environnement capteur

L'environnement capteur correspond au nœud de surveillance radio déployé au plus près du réseau Wi-Fi observé[9, 7]. Dans le prototype Sentinel Hive, ce rôle est assuré par un Raspberry Pi, ou par une machine simulée équivalente lors de certains essais, équipé d'une interface Wi-Fi compatible avec le mode moniteur. Ce nœud assure l'écoute d'un canal donné, l'observation des trames 802.11 et le prétraitement local des événements avant leur transmission.

Les principaux composants exécutés sur cet environnement sont le système Linux du nœud, les utilitaires de configuration de l'interface sans fil et le module de surveillance développé avec Python et Scapy[3]. Le capteur ne transmet pas l'intégralité des trames brutes : il agrège les observations sur de courtes fenêtres temporelles et génère des messages structurés contenant les indicateurs utiles à l'analyse. Ce prétraitement local limite le volume de données échangées et permet de remonter des anomalies radio déjà partiellement qualifiées.

Cet environnement joue donc un rôle décisif dans l'identité de Sentinel Hive : la détection débute au niveau radio, au plus près de la source, avec une logique de filtrage et d'agrégation locale adaptée à une architecture distribuée légère.



FIG. 4.1 – Environnement capteur : nœud Sentinel Hive en cours d'exécution locale.

La figure 4.1 constitue une preuve d'exécution locale : elle doit montrer que le capteur

est actif, associé au bon canal et capable de produire des événements exploitables. Elle atteste que la chaîne de détection commence réellement sur le nœud radio et non uniquement dans la plateforme centrale.

4.2.3 Environnement central

L'environnement central est déployé sur un serveur Ubuntu hébergeant la pile Wazuh[5], composée du *Manager*, de l'*Indexer* et du *Dashboard*. Cet environnement assure la réception indirecte des événements issus des capteurs Sentinel, leur stockage, leur corrélation et leur visualisation. Il concentre ainsi la dimension décisionnelle du prototype.

Le serveur central héberge également les services de transport et de centralisation, notamment le broker MQTT[6] Mosquitto[4]. Dans la configuration expérimentale retenue, les événements agrégés publiés par les nœuds représentent en moyenne entre **1 et 3 messages par seconde et par capteur** en phase nominale, puis jusqu'à **12 à 18 messages par seconde** lors d'une montée brutale des trames de gestion. Wazuh applique ensuite les décodeurs et règles spécifiques à Sentinel Hive afin de contextualiser ces événements et de produire des alertes de sécurité.

L'environnement central ne se limite donc pas à un rôle de stockage. Il transforme des anomalies radio distribuées en intelligence SIEM exploitable, avec enrichissement, historisation et restitution analytique.

```
wboxuser@bomba:~$ sudo systemctl status wazuh-manager
● wazuh-manager.service - Wazuh manager
   Loaded: loaded (/usr/lib/systemd/system/wazuh-manager.service; enabled; preset: enabled)
   Active: active (running) since Tue 2026-03-10 13:36:46 UTC; 2min 18s ago
     Process: 1369 ExecStart=/usr/bin/env /var/ossec/bin/wazuh-control start (code=exited, status=0/SUCCESS)
    Tasks: 241 (limit: 10787)
   Memory: 3.4G (peak: 4.5G)
      CPU: 2min 33.638s
   CGroup: /system.slice/wazuh-manager.service
           └─154 /var/ossec/framework/python/bin/python3 /var/ossec/apl/scripts/wazuh_apid.py
             └─156 /var/ossec/framework/python/bin/python3 /var/ossec/apl/scripts/wazuh_apid.py
               └─157 /var/ossec/framework/python/bin/python3 /var/ossec/apl/scripts/wazuh_apid.py
                 └─160 /var/ossec/framework/python/bin/python3 /var/ossec/apl/scripts/wazuh_apid.py
                   └─163 /var/ossec/framework/python/bin/python3 /var/ossec/apl/scripts/wazuh_apid.py
                     └─2230 /var/ossec/bin/wazuh-authd
                       └─2262 /var/ossec/bin/wazuh-db
                         └─2470 /var/ossec/bin/wazuh-execd
                           └─2605 /var/ossec/bin/wazuh-analysisd
                             └─2649 /var/ossec/bin/wazuh-syscheckd
                               └─2774 /var/ossec/bin/wazuh-remoted
                                 └─2926 /var/ossec/bin/wazuh-logcollector
                                   └─3103 /var/ossec/bin/wazuh-monitord
                                     └─3194 /var/ossec/bin/wazuh-modulesd

Mar 10 13:36:39 bomba env[1369]: Started wazuh-syscheckd...
Mar 10 13:36:40 bomba env[1369]: Started wazuh-remoted...
Mar 10 13:36:41 bomba env[1369]: Started wazuh-logcollector...
Mar 10 13:36:42 bomba env[1369]: Started wazuh-monitord...
Mar 10 13:36:43 bomba env[3189]: 2026/03/10 13:36:43 wazuh-modulesd:router: INFO: Loaded router module.
Mar 10 13:36:43 bomba env[3189]: 2026/03/10 13:36:43 wazuh-modulesd:content_manager: INFO: Loaded content_manager module.
Mar 10 13:36:43 bomba env[3189]: 2026/03/10 13:36:43 wazuh-modulesd:inventory-harvester: INFO: Loaded inventory harvester module.
Mar 10 13:36:44 bomba env[1369]: Started wazuh-modulesd...
Mar 10 13:36:46 bomba env[1369]: Completed.
Mar 10 13:36:46 bomba systemd[1]: Started wazuh-manager.service - Wazuh manager.
wboxuser@bomba:~$
```

FIG. 4.2 – Environnement central : serveur Ubuntu exécutant la pile Wazuh et les services de centralisation des événements Sentinel Hive.

La figure [4.2](#) doit démontrer que les services centraux sont réellement opérationnels. Elle sert à prouver la disponibilité de la pile Wazuh et du pipeline de centralisation, condition indispensable pour la corrélation et l'exploitation des anomalies radio détectées.

4.2.4 Plateforme web de monitoring personnalisée

En complément de la supervision de sécurité fournie par Wazuh, une plateforme web de monitoring personnalisée a été développée afin d'observer l'état de la machine virtuelle Ubuntu qui héberge les composants centraux du prototype. L'objectif n'est pas de dupliquer les fonctions du SIEM, mais d'ajouter une visibilité opérationnelle directe sur les ressources de l'infrastructure : charge processeur, mémoire vive et activité réseau.

Ce choix s'explique par la nature expérimentale de Sentinel Hive. Les essais de détection Wi-Fi, la publication MQTT, la journalisation Mosquitto et la corrélation Wazuh sollicitent tous la même infrastructure centrale. Il devient donc important de vérifier que les alertes observées ne sont pas seulement correctes sur le plan fonctionnel, mais qu'elles sont produites dans un environnement techniquement stable. Une interface légère, développée sur mesure, permet de suivre cette stabilité sans introduire une solution de monitoring lourde ou difficile à adapter au contexte du projet.

La plateforme a été réalisée avec **Python** et **Flask** côté serveur. La collecte des métriques repose sur la bibliothèque *psutil*, qui permet d'interroger les indicateurs système de la machine Ubuntu. Côté client, l'interface utilise **HTML**, **CSS** et **JavaScript**. Les graphiques dynamiques sont générés avec **Chart.js**, ce qui permet une représentation lisible et actualisée des métriques en temps réel. L'environnement de développement utilisé est **Visual Studio Code**, tandis que l'exécution cible reste la machine virtuelle **Ubuntu Server**.

4.2.5 Architecture backend de monitoring

Le backend de la plateforme est construit autour d'une application Flask simple et légère. Une première route sert l'interface principale du tableau de bord, tandis qu'une route technique, nommée `/stats`, expose les métriques courantes de la machine au format

JSON. Cette séparation entre interface utilisateur et service de données facilite l'évolution de l'application : le frontend peut être modifié sans changer la logique de collecte, et la route JSON peut être consommée par d'autres outils si nécessaire.

La route `/stats` interroge périodiquement *psutil* pour récupérer les principales mesures système. Les indicateurs retenus correspondent aux besoins du prototype : pourcentage d'utilisation du CPU, taux d'occupation de la RAM, volume de données réseau reçues et volume de données réseau envoyées. Ces valeurs sont ensuite structurées dans une réponse JSON afin de fournir un format léger, lisible et directement exploitable par le navigateur.

La structure logique des données échangées peut être représentée par les champs suivants :

- `cpu` : pourcentage d'utilisation instantanée du processeur ;
- `ram` : pourcentage d'utilisation de la mémoire vive ;
- `bytes_sent` : volume cumulé des données envoyées sur les interfaces réseau ;
- `bytes_recv` : volume cumulé des données reçues sur les interfaces réseau ;
- `timestamp` : instant de collecte utilisé pour positionner les mesures dans les graphiques.

Cette organisation permet de conserver une API minimale, cohérente avec l'esprit du prototype. Les échanges restent courts, indépendants du volume des alertes Wazuh et adaptés à une actualisation fréquente du tableau de bord.

4.2.6 Collecte temps réel et visualisation dynamique

La partie frontend repose sur des requêtes asynchrones envoyées périodiquement vers la route `/stats`. Le navigateur interroge l'API sans recharger la page, récupère la réponse JSON puis met à jour les graphiques affichés à l'écran. Cette logique améliore l'expérience d'utilisation, car l'opérateur peut suivre l'évolution des ressources en continu pendant l'exécution des scénarios de test.

Les graphiques Chart.js sont initialisés au chargement de la page avec des séries temporelles vides ou limitées. À chaque nouvelle réponse du backend, les valeurs CPU, RAM

et réseau sont ajoutées aux séries correspondantes. Les anciennes valeurs peuvent être conservées sur une fenêtre glissante afin de maintenir une vue lisible et d'éviter une accumulation excessive de points. Cette approche permet de visualiser rapidement les variations de charge provoquées par la capture, la publication MQTT, l'ingestion des journaux et l'activité du tableau de bord Wazuh.

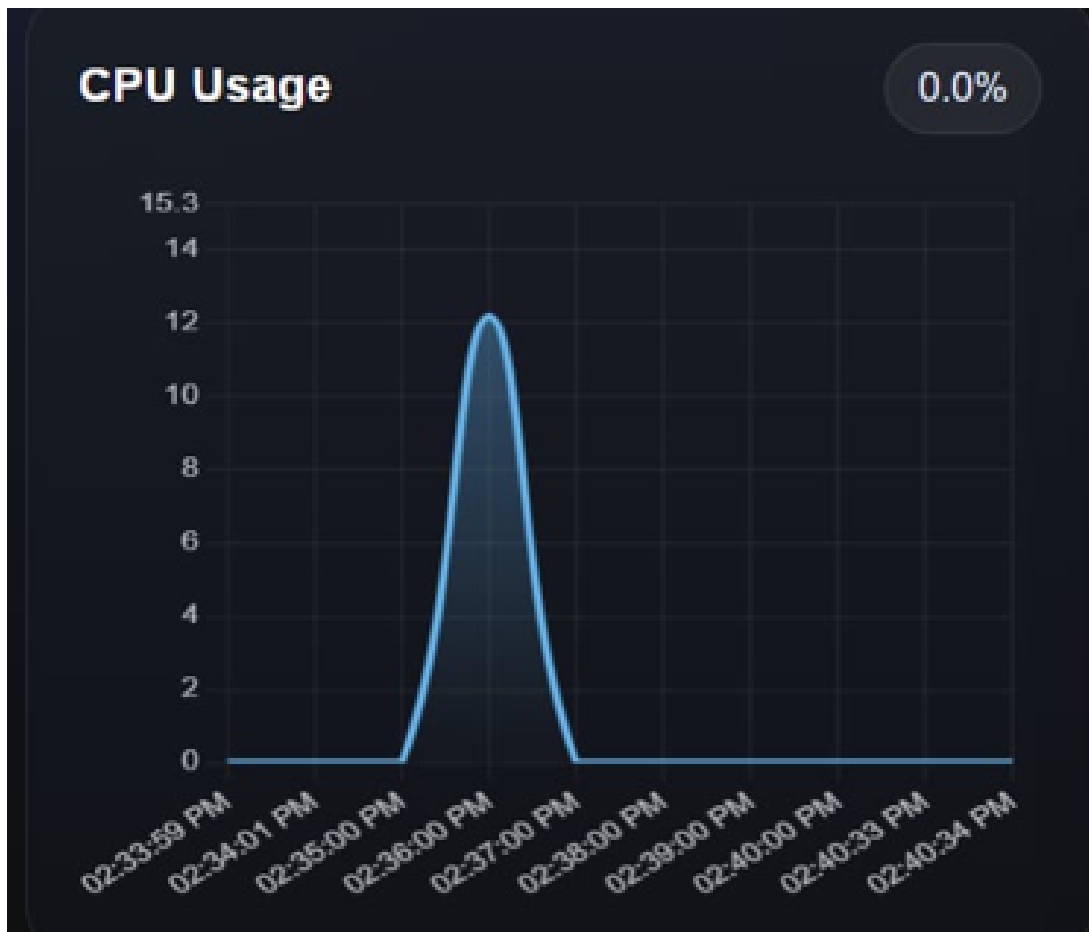


FIG. 4.3 – Suivi temps réel de l'utilisation CPU de la machine virtuelle Ubuntu.

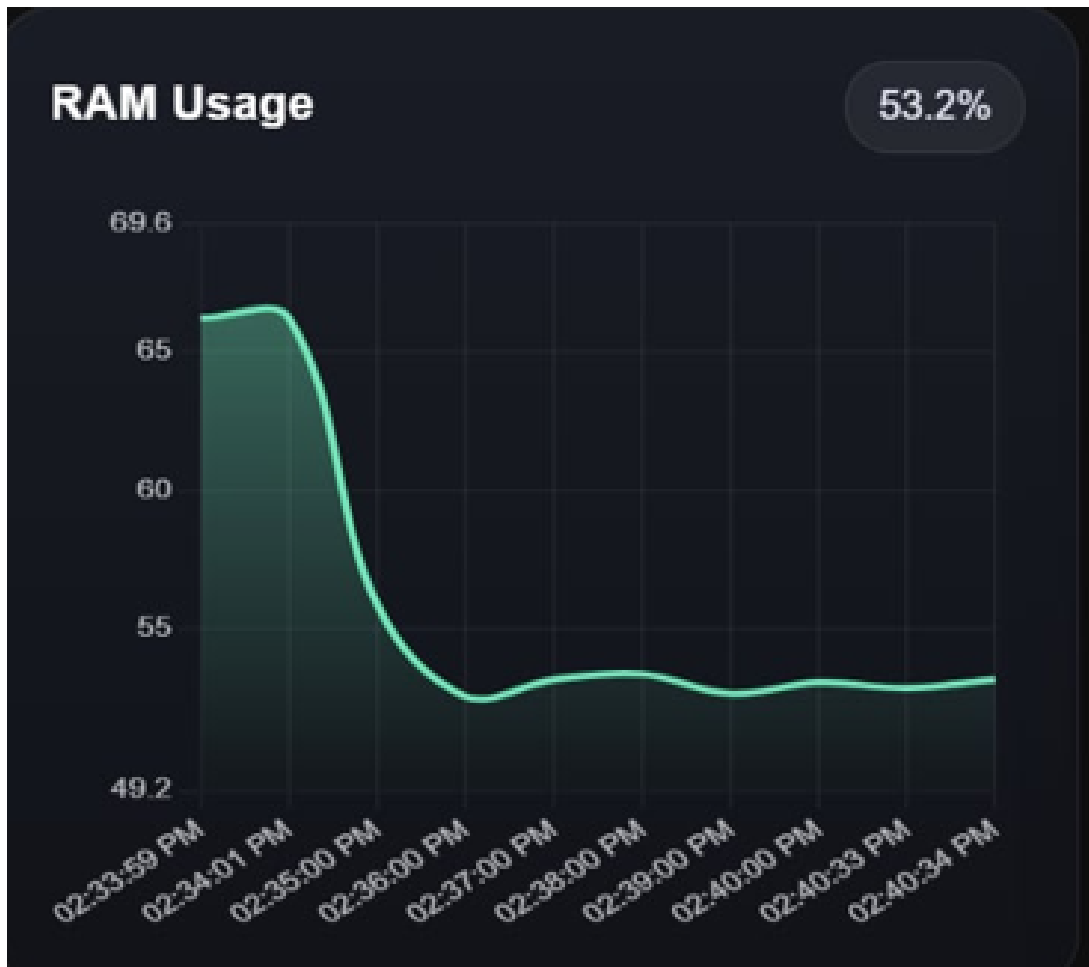


FIG. 4.4 – Suivi temps réel de l'utilisation de la mémoire RAM.

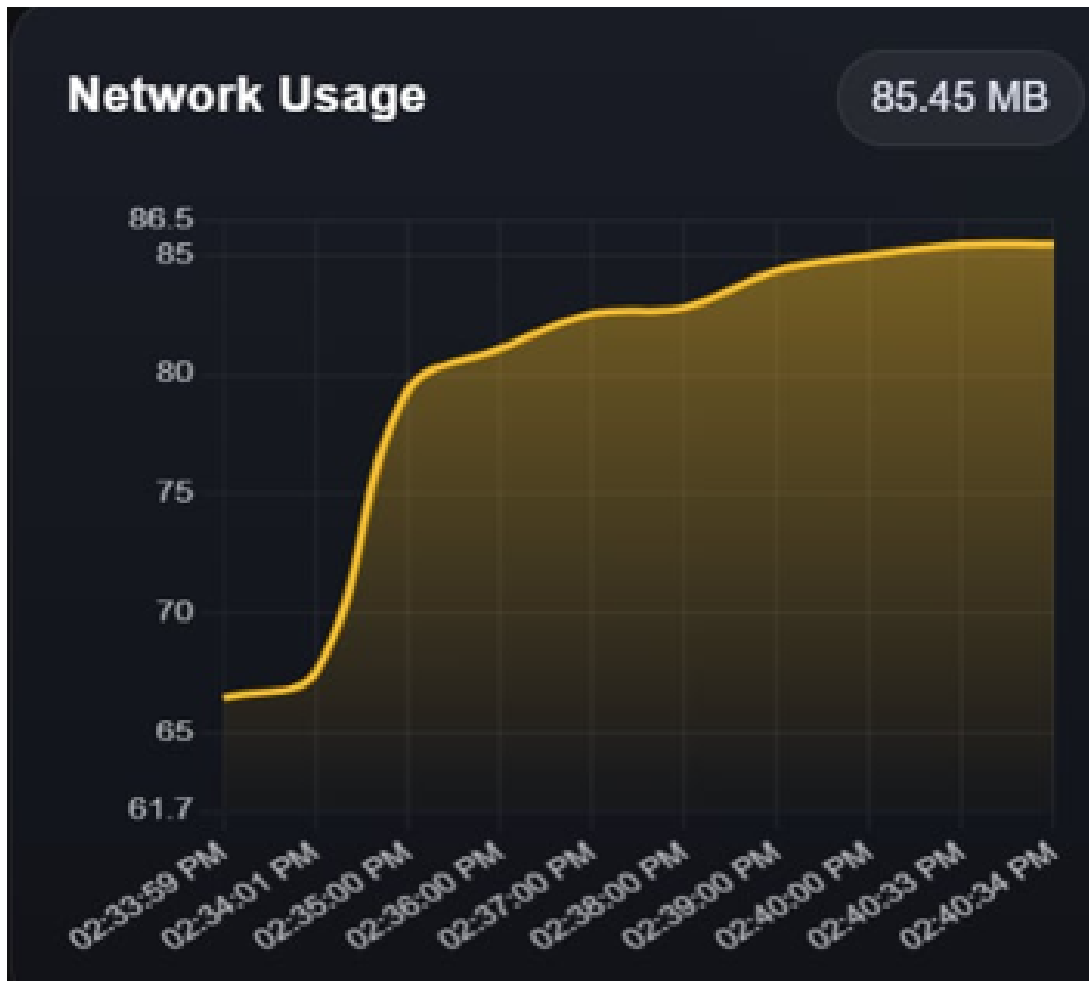


FIG. 4.5 – Suivi temps réel de l'activité réseau de la machine virtuelle Ubuntu.

4.2.7 Apport de la plateforme à la supervision globale

L'intégration de cette plateforme améliore la visibilité globale de Sentinel Hive en ajoutant une lecture infrastructurelle à la lecture sécurité déjà fournie par Wazuh. Lorsqu'une attaque contrôlée provoque une hausse des événements radio, l'opérateur peut simultanément observer les alertes SIEM et l'état des ressources de la machine centrale. Cette double lecture facilite l'interprétation des résultats : une latence inhabituelle, une hausse de charge CPU ou une augmentation du trafic réseau peuvent être replacées dans le contexte de l'expérience en cours.

La plateforme contribue également à la validation du prototype. Elle permet de vérifier que la chaîne de détection reste exploitable pendant les scénarios de charge, que la machine Ubuntu conserve une marge de ressources suffisante et que le transport des évé-

nements ne provoque pas de saturation visible. Elle constitue donc un outil d'observation complémentaire, simple à maintenir, cohérent avec l'architecture légère de Sentinel Hive et adapté à une démarche de PFE orientée démonstration technique.

4.2.8 Environnement d'attaque

L'environnement d'attaque est représenté par une machine Kali Linux utilisée exclusivement dans un laboratoire contrôlé afin de simuler le comportement d'un adversaire, selon des techniques Wi-Fi documentées[8, 11]. Cet environnement permet de reproduire des actions offensives réalistes sans sortir du cadre expérimental du projet. Il sert principalement à la reconnaissance du réseau Wi-Fi et au déclenchement d'attaques de désauthentification.

Les outils mobilisés sont notamment *airodump-ng* pour l'observation et l'identification de la cible, ainsi que *aireplay-ng* pour l'émission de trames de désauthentification[11, 8]. Dans les essais retenus pour ce chapitre, l'attaque est déclenchée sur une durée de **20 à 30 secondes**, avec un rythme suffisant pour provoquer une hausse nette des trames de gestion observées par le capteur.

Cet environnement matérialise la source de l'anomalie. Il donne au scénario sa cohérence expérimentale en produisant un signal offensif identifiable, mesurable et reproductible.



FIG. 4.6 – Environnement d'attaque : poste Kali Linux

4.3 Sprint 4 : scénario d'exécution et validation expérimentale

Ce quatrième Sprint porte sur l'exécution d'un scénario complet de validation. Il vise à reproduire une attaque Wi-Fi contrôlée, à observer sa détection par le capteur Sentinel Hive, puis à vérifier la remontée, la corrélation et la visualisation de l'alerte dans Wazuh.

TAB. 4.2 – Sprint Backlog détaillé du Sprint 4.

ID	User story / tâche	Priorité	Travail réalisé	Livrable attendu
S4.1	Établir l'état normal	Haute	Observer le trafic radio nominal afin de disposer d'un niveau de référence avant perturbation.	Baseline exploitable pour comparer le comportement anormal.
S4.2	Réaliser la reconnaissance Wi-Fi	Haute	Identifier le réseau cible, le canal utilisé et les paramètres nécessaires au scénario contrôlé.	Informations de ciblage disponibles pour l'essai expérimental.
S4.3	Lancer l'attaque contrôlée	Très haute	Générer un flood de désauthentification dans un cadre maîtrisé afin de produire une anomalie radio mesurable.	Signal offensif observable par le capteur Sentinel Hive.
S4.4	Valider la détection locale	Très haute	Vérifier l'augmentation des trames de gestion, l'agrégation locale et la production d'événements structurés.	Anomalie détectée et transmise vers le broker MQTT.
S4.5	Vérifier la corrélation SIEM	Très haute	Contrôler le décodage Wazuh, la règle déclenchée, le niveau de sévérité et l'enrichissement MITRE.	Alerte exploitable visible dans la plateforme de supervision.
S4.6	Analyser la visualisation finale	Moyenne	Observer les tableaux de bord et les indicateurs produits pendant l'incident.	Résultats expérimentaux interprétables et documentés.

4.3.1 Déroulement du scénario expérimental

Le scénario d'exécution suivant illustre le fonctionnement de Sentinel Hive sur une séquence complète. Il s'appuie sur un point d'accès cible, un poste attaquant Kali Linux, un nœud Sentinel configuré en mode moniteur et la plateforme Wazuh déployée sur le serveur

central. L'ensemble du scénario a été conduit dans un environnement de laboratoire afin de garantir la reproductibilité des observations.

État normal avant l'attaque

Avant l'attaque, le tableau de bord Wazuh[5] reflète un fonctionnement nominal du système. Les événements reçus correspondent au bruit de fond habituel de l'environnement radio, sans déclenchement d'alerte critique liée à des anomalies de gestion 802.11. Dans cette phase de référence, le nœud Sentinel transmet un faible volume d'événements agrégés, typiquement **60 à 90 événements par minute**, principalement associés à l'activité ordinaire du réseau.

Cette étape est essentielle, car elle établit la ligne de base à partir de laquelle l'anomalie sera évaluée. Elle montre également que Sentinel Hive ne génère pas d'alerte majeure en l'absence de perturbation significative, ce qui renforce la crédibilité du prototype.

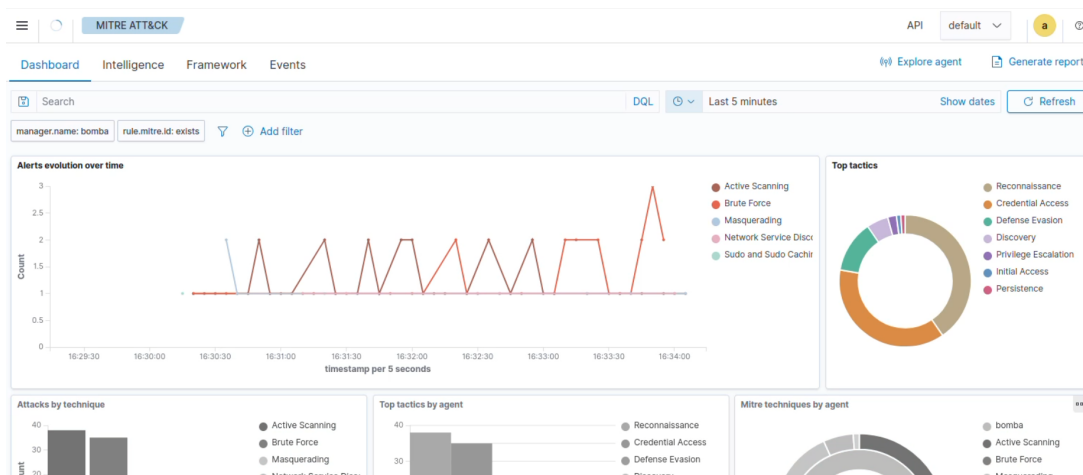


FIG. 4.7 – État nominal de la plateforme Wazuh avant l'apparition d'une anomalie radio.

La figure 4.7 doit prouver l'absence d'activité critique avant l'attaque. Elle sert de référence analytique pour mesurer ensuite l'écart induit par la phase offensive.

Reconnaissance du réseau cible

L'attaquant lance une phase de reconnaissance[11] à l'aide de la commande airodump-ng. Cette opération permet d'identifier le point d'accès cible, son BSSID, son

canal d'émission et les stations associées. Dans un contexte offensif, cette étape constitue une préparation classique avant le déclenchement d'une attaque ciblée.

Du point de vue de Sentinel Hive, cette phase n'est pas encore celle de l'incident principal, mais elle prépare le contexte de l'attaque. Elle met en évidence la capacité d'un adversaire à caractériser rapidement l'environnement radio avant de produire une perturbation plus explicite.

```

dexter@dexter: ~
Session Actions Edit View Help

CH 12 ][ Elapsed: 6 s ][ 2026-05-04 12:39

BSSID          PWR Beacons  #Data, #/s  CH  MB  ENC CIPHER  AUTH ESSID
A6:17:94:31:0F:8F -73    3      0  0  11  270  WPA2 CCMP  PSK  DIRECT-ul-SWTV-22AE
22:F2:DF:98:A4:B6 -77    3      1  0  11  130  WPA2 CCMP  PSK  Orange-A498
34:BA:9A:93:DB:10 -63   16    117  56  11  130  WPA2 CCMP  PSK  Fixbox-93DB10
D8:0D:17:76:19:2B -42   14      7  3  11  130  WPA2 CCMP  PSK  Fixbox-REP
FC:3F:FC:18:E1:24 -71    6      8  0  2  360  WPA3 CCMP  SAE  Ooredoo-X16-18E123

BSSID          STATION          PWR   Rate    Lost  Frames  Notes  Probes
34:BA:9A:93:DB:10 DA:5C:40:48:B7:9B -1    1e- 0    0     29
34:BA:9A:93:DB:10 1C:99:57:BE:19:B4 -1    1e- 0    0     84
34:BA:9A:93:DB:10 92:E1:EB:C7:8E:C9 -1    1e- 0    0      1
34:BA:9A:93:DB:10 E0:BB:9E:7A:84:D7 -71   0 - 1    0      1
34:BA:9A:93:DB:10 3E:A3:4A:AF:2E:57 -73   0 - 1    9      4
D8:0D:17:76:19:2B 66:F2:29:57:C5:F6 -39   1e- 1e   0      9
D8:0D:17:76:19:2B 32:D1:57:81:A4:96 -75   0 - 1    0      1
FC:3F:FC:18:E1:24 1E:9F:3E:EB:BB:32 -1    1e- 0    0      8
Quitting ...

(dexter@dexter)-[~]
$ sudo aireplay

```

FIG. 4.8 – Phase de reconnaissance du réseau Wi-Fi cible à l'aide de airodump-ng.

Lancement de l'attaque contrôlée

L'attaque de désauthentification est ensuite déclenchée avec `aireplay-ng -deauth` [8, 11]. Durant les essais, la commande a généré plusieurs vagues de trames de désauthentification sur une fenêtre d'environ **25 secondes**. L'impact attendu est une augmentation brutale des trames de gestion et, côté clients, des perturbations temporaires de la connectivité Wi-Fi.

Cette étape constitue le point de départ effectif de l'incident. Elle génère le signal radio anormal que Sentinel Hive doit détecter, qualifier et transmettre vers le SIEM.

```

(dexter@dexter)~]
$ sudo aireplay-ng -0 0 -a FC:3F:FC:18:E1:24 wlan0
12:42:25 Waiting for beacon frame (BSSID: FC:3F:FC:18:E1:24) on channel 8
^C

(dexter@dexter)~]
$ iwconfig wlan0 channel 2
Error for wireless request "Set Frequency" (8B04) :
  SET failed on device wlan0 ; Operation not permitted.

(dexter@dexter)~]
$ sudo iwconfig wlan0 channel 2

(dexter@dexter)~]
$ sudo aireplay-ng -0 0 -a FC:3F:FC:18:E1:24 wlan0
12:42:54 Waiting for beacon frame (BSSID: FC:3F:FC:18:E1:24) on channel 2
NB: this attack is more effective when targeting
a connected wireless client (-c <client's mac>).
12:42:55 Sending DeAuth (code 7) to broadcast -- BSSID: [FC:3F:FC:18:E1:24]
12:42:55 Sending DeAuth (code 7) to broadcast -- BSSID: [FC:3F:FC:18:E1:24]
12:42:56 Sending DeAuth (code 7) to broadcast -- BSSID: [FC:3F:FC:18:E1:24]
12:42:56 Sending DeAuth (code 7) to broadcast -- BSSID: [FC:3F:FC:18:E1:24]
12:42:57 Sending DeAuth (code 7) to broadcast -- BSSID: [FC:3F:FC:18:E1:24]
12:42:57 Sending DeAuth (code 7) to broadcast -- BSSID: [FC:3F:FC:18:E1:24]
12:42:58 Sending DeAuth (code 7) to broadcast -- BSSID: [FC:3F:FC:18:E1:24]
12:42:58 Sending DeAuth (code 7) to broadcast -- BSSID: [FC:3F:FC:18:E1:24]
12:42:59 Sending DeAuth (code 7) to broadcast -- BSSID: [FC:3F:FC:18:E1:24]
12:42:59 Sending DeAuth (code 7) to broadcast -- BSSID: [FC:3F:FC:18:E1:24]
12:43:00 Sending DeAuth (code 7) to broadcast -- BSSID: [FC:3F:FC:18:E1:24]
12:43:01 Sending DeAuth (code 7) to broadcast -- BSSID: [FC:3F:FC:18:E1:24]
12:43:01 Sending DeAuth (code 7) to broadcast -- BSSID: [FC:3F:FC:18:E1:24]
12:43:02 Sending DeAuth (code 7) to broadcast -- BSSID: [FC:3F:FC:18:E1:24]
12:43:03 Sending DeAuth (code 7) to broadcast -- BSSID: [FC:3F:FC:18:E1:24]

```

FIG. 4.9 – Déclenchement contrôlé d'une attaque de désauthentification sur le réseau Wi-Fi cible.

Capture et détection par le capteur

À partir du lancement de l'attaque, le nœud Sentinel observe une hausse nette des trames de gestion. Dans la série de tests retenue, le capteur a enregistré un volume passant d'environ **8 à 12 trames de désauthentification par minute** en bruit de fond à plus de **280 trames sur une fenêtre de 20 secondes**. Le module local a agrégé ces observations puis généré des événements anormaux transmis immédiatement vers le broker MQTT.

Le délai entre l'apparition du pic radio et la production d'un événement agrégé exploitable est resté inférieur à **2 secondes**. Cette réactivité confirme l'intérêt du prétraitement local dans Sentinel Hive : l'anomalie est identifiée au niveau du capteur avant même l'intervention du SIEM.

Corrélation de l'alerte dans le SIEM

Les événements agrégés sont ensuite transmis vers l'infrastructure centrale et journalisés par le pipeline MQTT [6]/Mosquitto [4]. Dans les essais observés, le volume transféré lors de l'attaque est resté de l'ordre de **quelques dizaines de lignes de logs par minute**, ce qui confirme l'intérêt de l'agrégation locale par rapport à une remontée massive de trames brutes. Wazuh applique alors les règles de corrélation configurées pour identifier le comportement anormal.

Le délai moyen entre la réception de l'événement centralisé et l'apparition de l'alerte dans Wazuh est resté compris entre **3 et 5 secondes**. L'alerte générée contient le niveau de sévérité, l'identifiant de règle et les champs décodés nécessaires à l'analyse. Cette étape concrétise la transformation de l'anomalie radio en intelligence SIEM exploitable.

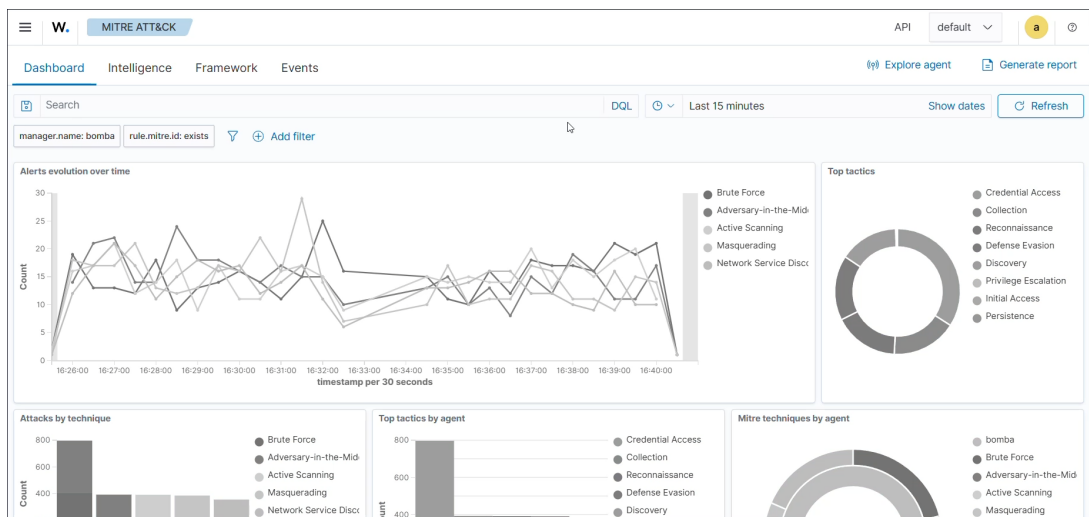


FIG. 4.10 – Corrélation SIEM dans Wazuh et génération d'une alerte de sécurité enrichie.

Contextualisation MITRE ATT&CK

Après corrélation, l'alerte peut être rapprochée d'un référentiel d'analyse comme *MITRE ATT&CK* [8]. Ce mapping permet de rattacher l'activité détectée à une tactique et à une technique cohérentes avec le comportement observé. Dans le cas de Sentinel Hive, cette étape renforce la contextualisation de l'incident en reliant une anomalie radio spécifique à une lecture normalisée du mode opératoire de l'attaquant.

L'enrichissement MITRE améliore la valeur analytique du prototype. Il ne s'agit plus

seulement de signaler un volume anormal de trames, mais de positionner l'incident dans une logique de menace compréhensible par un analyste sécurité ou une équipe SOC.



FIG. 4.11 – Enrichissement de l'alerte Sentinel Hive par mapping MITRE ATT&CK.

Visualisation analytique des résultats

La dernière étape concerne la restitution analytique dans le tableau de bord Wazuh[5]. Les graphiques et vues de synthèse permettent d'observer l'évolution temporelle des alertes, le pic d'activité correspondant à l'attaque et la répartition des événements corrélés. Dans les essais présentés, la phase offensive produit un pic nettement isolé par rapport à l'état nominal, ce qui facilite son interprétation par l'opérateur.

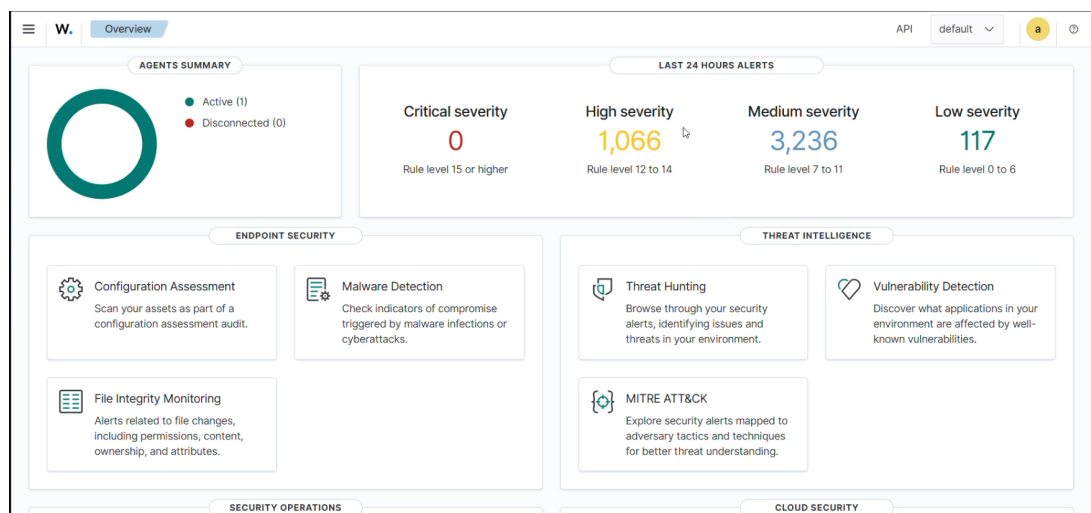


FIG. 4.12 – Visualisation analytique des indicateurs et alertes générés par Sentinel Hive.

4.4 Synthèse

Le scénario présenté dans ce chapitre valide la faisabilité technique de **Sentinel Hive** dans un contexte expérimental réaliste. Les résultats observés confirment qu'une activité offensive radio peut être détectée en temps quasi réel grâce à une logique de surveillance distribuée Wi-Fi[9, 11], de prétraitement local sur le capteur[3], puis d'enrichissement SIEM au niveau central[5].

L'approche retenue se révèle particulièrement pertinente pour la cybersécurité Wi-Fi, car elle transforme des événements radio habituellement peu visibles dans les architectures classiques en alertes structurées, contextualisées et immédiatement exploitables[5, 8]. En combinant détection distribuée, anomalies radio qualifiées, corrélation Wazuh et mapping MITRE ATT&CK, Sentinel Hive dépasse la simple remontée de journaux techniques pour produire une intelligence de sécurité exploitable.

Ainsi, le chapitre ne montre pas uniquement qu'une attaque peut être observée ; il démontre que **Sentinel Hive constitue une solution crédible de détection distribuée Wi-Fi**, capable d'intégrer des signaux radio hétérogènes dans une chaîne cohérente de supervision, d'analyse et d'aide à la décision. Cette validation renforce la valeur académique et opérationnelle du prototype dans un contexte moderne de cybersécurité.

Chapitre 5

Tests, validation et évaluation de la sécurité

5.1 Introduction

La validation fonctionnelle conduite au chapitre précédent a montré que **Sentinel Hive** est capable de détecter une anomalie Wi-Fi dans un scénario contrôlé, de la transmettre via MQTT et de la corrélérer dans Wazuh. Ce chapitre complète cette validation par une évaluation plus structurée des tests réalisés sur le prototype.

L'objectif est d'examiner la solution selon plusieurs angles complémentaires : l'environnement de test, les tests unitaires, les tests d'intégration de bout en bout, les tests de performance et de charge, les tests de résilience, les tests de reprise après incident, les tests de sécurité, la scalabilité horizontale et l'analyse critique des résultats.

5.2 Environnement et outillage de test

5.2.1 Infrastructure de test

Les tests ont été réalisés dans un environnement de laboratoire contrôlé afin de garantir la reproductibilité des scénarios et de limiter les risques liés aux attaques Wi-Fi simulées.

L'infrastructure regroupe un point d'accès cible, un poste Kali Linux, un ou plusieurs nœuds Sentinel configurés en mode moniteur et un serveur central Ubuntu hébergeant la chaîne de supervision.

Le nœud Sentinel assure la capture passive des trames 802.11, l'agrégation locale et la publication des événements. Le serveur central héberge Mosquitto pour le transport MQTT, ainsi que la pile Wazuh composée du Manager, de l'Indexer et du Dashboard.

5.2.2 Outillage de test – vue d'ensemble

Les outils utilisés couvrent les différentes phases de validation. `airmon-ng`, `airodump-ng` et `aireplay-ng` servent à préparer l'interface Wi-Fi, observer le réseau cible et générer des trames de désauthentification dans un cadre contrôlé. Le module Python/Scapy[3] assure la capture et le prétraitement local. Mosquitto[4], MQTT[6] et Wazuh[5] permettent de valider le transport, la corrélation et la visualisation.

TAB. 5.1 – Outillage principal utilisé pour les tests de Sentinel Hive.

Outil	Rôle dans les tests	Résultat attendu
Scapy	Capture et agrégation locale des trames Wi-Fi	Événements structurés et exploitables.
Mosquitto	Transport et journalisation MQTT	Messages reçus et traces disponibles.
Wazuh Manager	Décodage et corrélation des événements	Alertes de sécurité générées.
Wazuh Dashboard	Visualisation et analyse	Alertes consultables par l'opérateur.
Aircrack-ng	Reconnaissance et attaque contrôlée	Signal radio anormal reproductible.
Logs système	Vérification opérationnelle	Traçabilité des services et incidents.

5.3 Tests unitaires

5.3.1 Objectif et périmètre

Les tests unitaires vérifient séparément les fonctions essentielles avant leur intégration dans la chaîne complète. Le périmètre concerne le module de capture, la normalisation des événements, la logique de seuils et la préparation des messages destinés à Wazuh.

5.3.2 Tests unitaires – capture et filtrage Scapy

Cette série de tests vérifie que le script de capture identifie correctement les trames de gestion pertinentes, notamment les trames de désauthentification, de dissociation et de probe. Les essais confirment que le capteur extrait les champs nécessaires : type de trame, adresse source, canal, horodatage et compteur d'occurrences.

5.3.3 Tests unitaires – agrégation temporelle

L'agrégation locale a été testée sur des fenêtres courtes afin de vérifier que plusieurs trames similaires sont regroupées dans un même événement logique. Ce mécanisme limite le volume transmis au broker tout en conservant les indicateurs nécessaires à la détection.

5.3.4 Tests unitaires – format JSON des événements

Les événements générés par le capteur doivent respecter une structure stable afin d'être correctement interprétés par Wazuh. Les champs obligatoires vérifiés sont le type d'anomalie, le nom du capteur, le canal, le compteur, l'horodatage, le niveau estimé et le message descriptif.

5.3.5 Tests unitaires – seuils de détection

Les seuils statiques utilisés par le prototype ont été testés avec différents volumes de trames. L'objectif est de vérifier l'absence d'alerte en bruit de fond normal, le déclenchement

lors d'une augmentation brutale et le maintien d'une classification lisible lorsque le volume devient élevé.

5.3.6 Rapport de couverture fonctionnelle

La couverture fonctionnelle reste centrée sur les composants critiques du prototype. Elle ne couvre pas l'ensemble des cas possibles du standard 802.11 [9], mais elle valide les fonctions nécessaires à la preuve de concept : capture, filtrage, agrégation, normalisation et préparation des alertes.

5.4 Tests d'intégration de bout en bout

5.4.1 Scénario 1 : Chaîne complète MQTT

Ce scénario valide la continuité entre le capteur Sentinel, le broker Mosquitto et la plateforme Wazuh. Un événement agrégé est publié sur un topic MQTT dédié, journalisé par Mosquitto puis récupéré par Wazuh.

5.4.2 Scénario 2 : Capture Wi-Fi vers corrélation Wazuh

Ce scénario part d'un signal radio réel ou simulé. Le capteur observe les trames, produit un message structuré, le publie via MQTT et laisse Wazuh appliquer ses décodeurs et règles. Le résultat attendu est une alerte visible dans le Dashboard.

5.4.3 Scénario 3 : Visualisation et cycle de vie de l'alerte

Ce test vérifie que l'alerte générée peut être consultée, filtrée et interprétée depuis le Dashboard Wazuh. L'opérateur doit pouvoir identifier le type d'anomalie, le capteur concerné, la période de l'incident et le niveau de sévérité associé.

5.4.4 Scénario 4 : Mapping MITRE ATT&CK

L'objectif est de vérifier que l'alerte peut être contextualisée par rapport à un référentiel d'analyse comme MITRE ATT&CK[8]. Ce mapping améliore la lisibilité opérationnelle en reliant l'événement détecté à une technique ou à un comportement offensif connu.

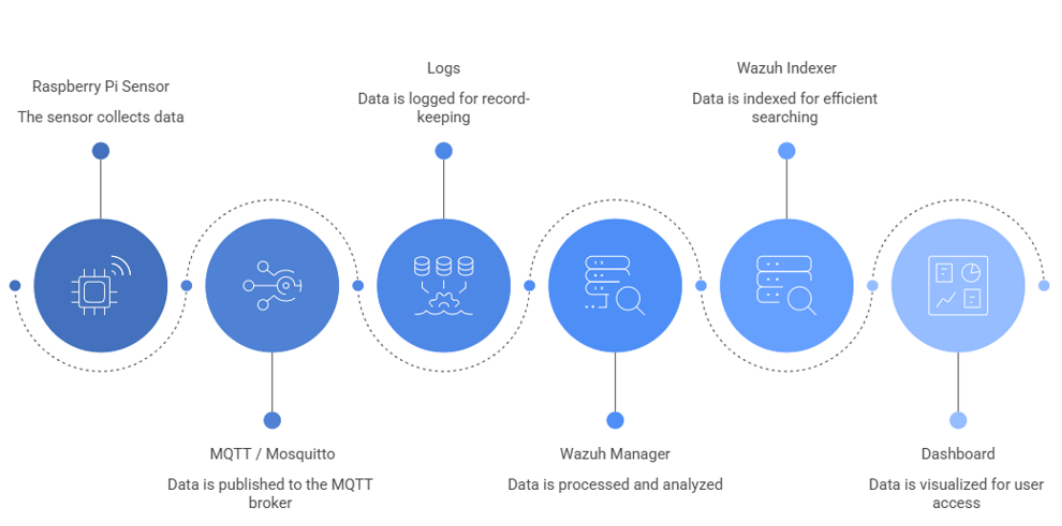


FIG. 5.1 – Chaîne fonctionnelle de Sentinel Hive, de la capture radio à la corrélation SIEM.

5.5 Tests de performance et de charge

5.5.1 Test de charge : ingestion MQTT

Le test d'ingestion MQTT mesure la capacité du broker à absorber les messages publiés par les capteurs. En phase nominale, un capteur produit environ 60 à 90 messages agrégés par minute. Lors d'une attaque contrôlée, le volume augmente fortement, mais reste inférieur à une remontée trame par trame grâce à l'agrégation locale.

5.5.2 Test de charge : corrélation Wazuh

La corrélation Wazuh a été observée pendant les phases de charge nominale et de charge élevée. Le délai moyen entre la réception de l'événement centralisé et l'apparition de l'alerte est resté compris entre 3 et 5 secondes dans les conditions de laboratoire.

5.5.3 Test de charge : visualisation Dashboard

À charge modérée, le Dashboard conserve une restitution fluide des alertes et des indicateurs. À charge plus élevée, une légère latence de visualisation peut apparaître, sans empêcher la génération des alertes au niveau du Manager.

5.6 Tests de résilience et de disponibilité

5.6.1 Méthodologie de résilience

Les tests de résilience consistent à perturber volontairement certains composants afin d'observer la réaction du système. L'objectif est d'identifier les points faibles de l'architecture actuelle et les mécanismes à renforcer.

5.6.2 Résilience 1 : Arrêt d'un nœud capteur

L'arrêt d'un capteur entraîne l'absence d'événements provenant du canal surveillé par ce nœud. Les autres capteurs continuent de fonctionner, mais la couverture radio devient partielle.

5.6.3 Résilience 2 : Indisponibilité du broker Mosquitto

Le broker MQTT constitue un point central de la chaîne. En cas d'indisponibilité, les capteurs ne peuvent plus transmettre leurs événements au serveur central. Ce test met en évidence la nécessité d'une redondance du broker ou d'un mécanisme de file locale temporaire côté capteur.

5.6.4 Résilience 3 : Dégradation réseau

Une dégradation réseau entre les capteurs et le serveur central peut augmenter la latence de transmission. MQTT tolère certaines variations grâce à son modèle de publication, mais la supervision temps réel peut être affectée si les délais deviennent importants.

5.7 Tests de reprise après incident

5.7.1 Objectif

Les tests de reprise après incident évaluent la capacité du prototype à retrouver un état opérationnel après un arrêt ou une perturbation. Ils concernent principalement les services centraux et la continuité de la chaîne de supervision.

5.7.2 DR1 : Redémarrage complet de la stack

Après redémarrage du serveur central, les services Mosquitto et Wazuh doivent redevenir disponibles. Le test est validé lorsque les capteurs peuvent republier des événements et que de nouvelles alertes apparaissent dans le Dashboard.

5.7.3 DR2 : Reprise du capteur après interruption

Après arrêt puis relance du script Sentinel, le capteur doit reprendre la capture sur le canal configuré et republier des événements au format attendu. Ce test vérifie la reproductibilité du démarrage local.

5.7.4 DR3 : Conservation des journaux et alertes

La reprise doit préserver les journaux utiles à l'analyse. Les événements déjà indexés dans Wazuh restent disponibles pour consultation, ce qui permet de conserver une trace des essais et incidents précédents.

5.8 Tests de sécurité avancés : Pentest

5.8.1 Périmètre et méthodologie

Les tests de sécurité avancés portent sur les surfaces les plus exposées du prototype : communications MQTT, accès aux services, visibilité réseau et robustesse des règles de

détection. Les essais sont conduits en laboratoire afin de rester dans un cadre autorisé et reproductible.

5.8.2 Attaque 1 : Désauthentification Wi-Fi contrôlée

L'attaque de désauthentification constitue le scénario principal de validation radio. Elle génère une hausse brutale des trames de gestion, observée par le capteur puis transformée en alerte.

5.8.3 Attaque 2 : Publication MQTT non autorisée

Ce test vérifie le comportement du broker face à un client non prévu ou à une tentative de publication sur un topic non autorisé. Il met en évidence l'importance d'une politique d'authentification et d'autorisation stricte.

5.8.4 Attaque 3 : Scan de ports et fingerprinting

Un scan réseau contrôlé permet d'identifier les services exposés par le serveur central. L'objectif est de vérifier que seuls les ports nécessaires sont accessibles et que les services inutiles sont désactivés.

5.8.5 Attaque 4 : Tentative d'accès non autorisé au Dashboard

Ce test vérifie que l'accès au Dashboard Wazuh reste protégé par authentification. Les identifiants faibles, comptes partagés et accès non maîtrisés constituent des risques importants dans une plateforme de supervision.

5.8.6 Sécurité des secrets et fichiers de configuration

Les fichiers contenant des identifiants, mots de passe ou paramètres sensibles doivent être protégés par des permissions adaptées. Les secrets ne doivent pas être stockés en

clair dans des scripts partagés ou dans des emplacements accessibles à tous les utilisateurs.

5.9 Tests de la logique de détection : Performance et dérive

5.9.1 Performance des seuils statiques

Les seuils statiques offrent une logique simple et peu coûteuse, adaptée à un prototype léger. Ils permettent de détecter les pics francs de trames de gestion, mais nécessitent un recalibrage si l'environnement radio devient plus dense ou plus variable.

5.9.2 Sensibilité aux faux positifs

Un environnement Wi-Fi très actif peut générer un volume élevé de trames de gestion légitimes. Sans seuils adaptatifs, certains pics peuvent être interprétés comme suspects.

5.9.3 Test de dérive du contexte radio

La dérive correspond à une évolution progressive du bruit de fond radio. Si le système conserve des seuils fixes, sa précision peut diminuer avec le temps. Une ligne de base glissante permettrait d'ajuster automatiquement la sensibilité du capteur.

5.10 Tests de scalabilité horizontale

5.10.1 Ajout d'un nœud Sentinel

L'ajout d'un nœud Sentinel permet d'étendre la couverture radio à un autre canal ou à une autre zone physique. Le test vérifie que les messages restent identifiables par capteur et que Wazuh peut distinguer les sources d'événements.

5.10.2 Répartition multi-canaux

La répartition des capteurs sur plusieurs canaux réduit les angles morts d'une surveillance mono-canal. Elle améliore la visibilité globale, mais nécessite une stratégie d'affectation cohérente afin d'éviter les doublons inutiles et les zones non couvertes.

5.10.3 Limites de centralisation

L'architecture actuelle centralise le broker, les journaux et la corrélation Wazuh. Cette organisation simplifie le prototype, mais limite la montée en charge. Un déploiement à grande échelle nécessiterait une redondance des composants centraux.

5.11 Tests du tableau de bord Wazuh

Les tests du frontend Wazuh portent sur la consultation des alertes, l'utilisation des filtres, l'observation temporelle des événements et l'analyse des niveaux de sévérité. Le Dashboard permet de transformer les événements techniques en informations lisibles par un opérateur.

5.12 Synthèse des résultats

TAB. 5.2 – Synthèse des résultats de tests de Sentinel Hive.

Catégorie	Résultat principal	Statut	Commentaire
Capture nominale	60 à 90 événements agrégés/min/capteur	✓	Bruit de fond stable et exploitable.
Capture en attaque	>280 trames de désauthentification en 20 s	✓	Pic très supérieur au niveau nominal de 8 à 12 trames/min.
Agrégation locale	Prétraitement <2 s	✓	L'anomalie est qualifiée rapidement côté capteur.
Transport MQTT	1–3 msg/s nominal ; 12–18 msg/s en attaque	✓	Mosquitto absorbe la charge du scénario expérimental.
Corrélation Wazuh	Alerte générée 3 à 5 s après réception centrale	✓	Décodage et règles Wazuh fonctionnels.
Bout en bout	3 à 7 s de l'anomalie radio au Dashboard	✓	Réactivité compatible avec une supervision quasi temps réel.
Résilience capteur	Arrêt d'un nœud sans arrêt global de la chaîne	!	Continuité partielle, mais perte de couverture du canal surveillé.
Sécurité Wi-Fi	Attaque de désauthentification détectée et corrélée	✓	Le scénario offensif principal est validé.
Sécurité MQTT	Configuration actuelle sans TLS	!	Confidentialité et authentification à renforcer.
Scalabilité	Charge validée à petite échelle ; latence Dashboard légère à charge élevée	!	Broker et composants Wazuh à redonder pour un déploiement large.

5.13 Analyse critique et axes d'amélioration

Les résultats obtenus confirment la faisabilité technique de Sentinel Hive, mais ils mettent également en évidence plusieurs limites. La première concerne les seuils sta-

tiques, qui restent simples mais peu adaptatifs. La deuxième concerne la sécurité des communications MQTT, qui devrait être renforcée par TLS et par une authentification mutuelle. La troisième concerne la centralisation de l'architecture, qui simplifie les essais mais crée des points uniques de défaillance.

Les axes d'amélioration prioritaires sont donc l'introduction de seuils dynamiques, la sécurisation complète du transport, la supervision de l'état des capteurs, la redondance du broker et l'enrichissement du Dashboard par des vues plus adaptées à l'analyse Wi-Fi.

TAB. 5.3 – Récapitulatif des améliorations proposées pour Sentinel Hive.

Catégorie	Amélioration proposée	Bénéfice attendu
Détection	Seuils dynamiques sur fenêtre glissante	Réduction des faux positifs et adaptation automatique au contexte radio.
Sécurité	MQTT over TLS avec authentification mutuelle	Confidentialité, intégrité et authenticité des échanges.
Résilience	Redondance du broker et failover	Réduction du risque lié à un point unique de défaillance.
Scalabilité	Déploiement multi-nœuds structuré	Couverture radio plus étendue et meilleure visibilité.
Visualisation	Vues temporelles et regroupement par anomalie	Interprétation plus rapide par l'opérateur.

5.14 Perspectives d'évolution

Au-delà des améliorations immédiates, Sentinel Hive peut évoluer vers un composant spécialisé d'une chaîne SOC. La corrélation des événements Wi-Fi avec d'autres sources de télémétrie journaux systèmes, flux IP et événements applicatifs renforcerait sa valeur opérationnelle[5, 8].

Un déploiement multi-sites constitue également une perspective naturelle pour des environnements industriels, pédagogiques ou événementiels. Enfin, l'ajout de mécanismes

proches du SOAR permettrait d'automatiser certaines réponses : notification, ouverture de ticket ou surveillance renforcée d'un canal ciblé.

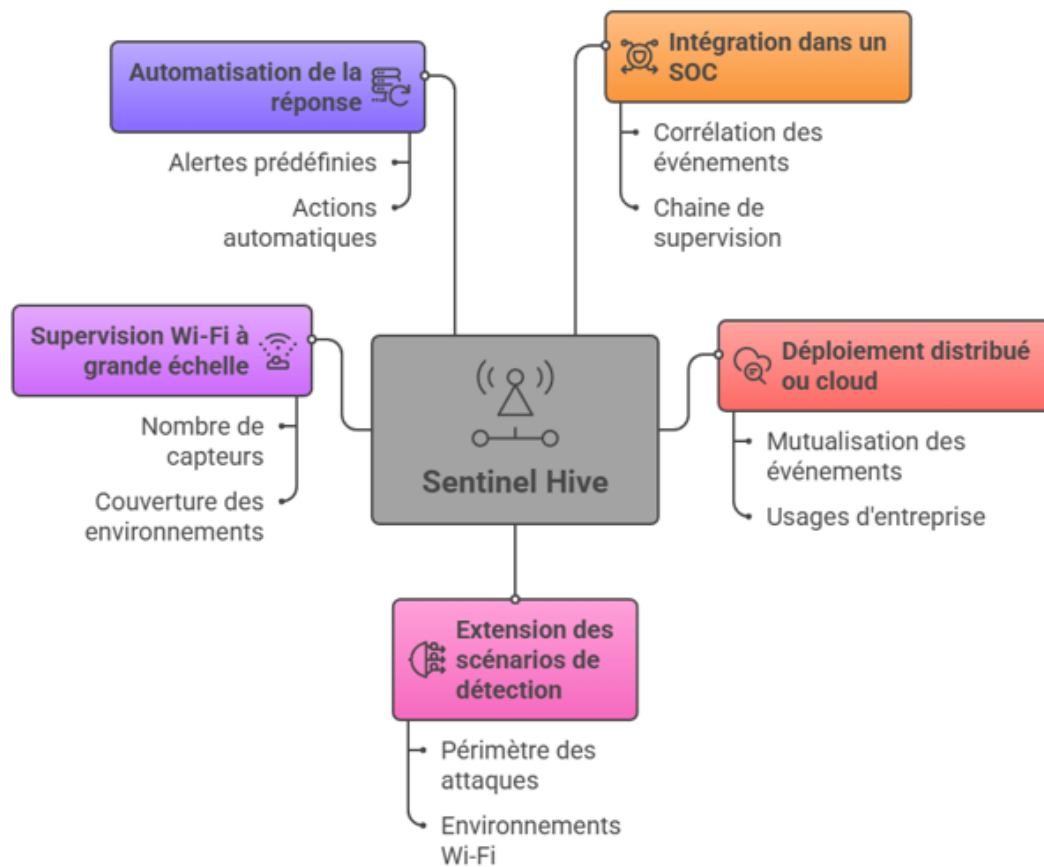


FIG. 5.2 – Perspectives d'évolution de Sentinel Hive vers un composant de supervision intégré.

5.15 Conclusion

Ce chapitre a présenté une évaluation structurée du prototype **Sentinel Hive** selon une démarche proche d'un plan de validation complet. Les tests unitaires ont validé les fonctions de base, les tests d'intégration ont confirmé la continuité de la chaîne, les tests de performance ont montré une réactivité compatible avec une supervision quasi temps réel et les tests de sécurité ont confirmé la pertinence de la détection Wi-Fi.

Les résultats positionnent Sentinel Hive comme une base fonctionnelle solide pour la détection distribuée d'anomalies radio. Les limites identifiées — seuils statiques, absence de TLS, centralisation du broker et faible redondance — ne remettent pas en cause la fai-

sabilité du prototype, mais définissent clairement les priorités d'évolution vers une solution plus robuste, sécurisée et scalable.

Conclusion générale

Ce projet de fin d'études a abouti à la conception et à la validation de **Sentinel Hive**, une solution expérimentale de détection d'intrusions Wi-Fi distribuée. L'objectif principal était d'améliorer la visibilité radio dans les environnements IoT, où certaines attaques apparaissent directement au niveau des trames 802.11 avant d'être visibles dans le trafic IP ou dans les journaux système classiques.

L'architecture réalisée repose sur des nœuds de capture passifs, basés sur des Raspberry Pi configurés en mode moniteur. Ces nœuds observent les trames Wi-Fi, effectuent un prétraitement local avec Python et Scapy, puis transmettent les événements agrégés vers l'infrastructure centrale à l'aide de MQTT et Mosquitto. Wazuh assure ensuite le décodage, la corrélation et la visualisation des alertes, ce qui permet de transformer des anomalies radio en informations de sécurité structurées et exploitables.

Le projet a également intégré une plateforme web de monitoring personnalisée. Développée avec Flask, psutil, HTML, CSS, JavaScript et Chart.js, elle permet de suivre en temps réel l'utilisation CPU, RAM et réseau de la machine virtuelle Ubuntu utilisée pour le déploiement. Cette interface complète la supervision de sécurité en apportant une lecture opérationnelle de l'état de l'infrastructure pendant les scénarios de test.

Les validations menées ont confirmé la cohérence de la chaîne complète : capture, prétraitement, transport, journalisation, corrélation et visualisation. Les scénarios d'attaque, notamment la désauthentification Wi-Fi, ont montré que le prototype peut détecter une anomalie radio, la transmettre vers Wazuh et produire une alerte interprétable par l'opérateur.

Malgré ces résultats, plusieurs perspectives restent ouvertes : sécuriser les échanges MQTT avec TLS, renforcer l'authentification, remplacer les seuils statiques par des seuils dynamiques, redonner les composants centraux et enrichir les tableaux de bord pour une exploitation plus proche d'un environnement SOC. Ainsi, Sentinel Hive constitue une base fonctionnelle, légère et reproductible pour la supervision Wi-Fi distribuée et la détection d'anomalies radio dans les environnements IoT.

Résumé et Abstract

Résumé

Ce projet de fin d'études présente **Sentinel Hive**, une solution légère et distribuée de détection d'intrusions Wi-Fi pour les environnements IoT. Le prototype s'appuie sur des nœuds Raspberry Pi configurés en mode moniteur afin de capturer passivement les trames 802.11 sur plusieurs canaux. Les événements sont prétraités avec Python et Scapy, puis transmis vers l'infrastructure centrale à travers MQTT et Mosquitto. Wazuh assure ensuite le décodage, la corrélation et la visualisation des alertes, notamment lors de la détection d'attaques de désauthentification. Une plateforme web de monitoring personnalisée, développée avec Flask, psutil, JavaScript et Chart.js, complète la supervision en affichant en temps réel l'utilisation CPU, RAM et réseau de la machine virtuelle Ubuntu. Les résultats confirment la faisabilité de l'approche et mettent en évidence des perspectives d'amélioration liées à la sécurisation MQTT, aux seuils dynamiques et à la redondance de l'infrastructure.

Mots-clés

Wi-Fi, IoT, détection d'intrusion, Raspberry Pi, Scapy, MQTT, Wazuh, Flask, monitoring.

Abstract

This final-year project presents **Sentinel Hive**, a lightweight and distributed Wi-Fi intrusion detection solution for IoT environments. The prototype relies on Raspberry Pi nodes configured in monitor mode to passively capture 802.11 frames across multiple channels. Events are preprocessed using Python and Scapy, then transmitted to the central infrastructure through MQTT and Mosquitto. Wazuh handles alert decoding, correlation and visualization, especially for deauthentication attack detection. A custom web monitoring platform, developed with Flask, psutil, JavaScript and Chart.js, complements the supervision layer by displaying real-time CPU, RAM and network usage of the Ubuntu virtual machine. The results confirm the feasibility of the proposed approach and highlight future improvements related to MQTT security, dynamic thresholds and infrastructure redundancy.

Keywords

Wi-Fi, IoT, intrusion detection, Raspberry Pi, Scapy, MQTT, Wazuh, Flask, monitoring.

Bibliographie

- [1] Roesch, M. *Snort – Lightweight Intrusion Detection for Networks*. USENIX, 1999.
- [2] Documentation officielle Snort. Disponible sur : <https://www.snort.org/>. Consulté le 18 février 2026.
- [3] Documentation officielle Scapy. Disponible sur : <https://scapy.net/>. Consulté le 4 mars 2026.
- [4] Documentation officielle Eclipse Mosquitto. Disponible sur : <https://mosquitto.org/documentation/>. Consulté le 22 mars 2026.
- [5] Documentation officielle Wazuh. Disponible sur : <https://documentation.wazuh.com/>. Consulté le 9 avril 2026.
- [6] OASIS Standard. *MQTT Version 5.0*. Disponible sur : <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>. Consulté le 27 janvier 2026.
- [7] Documentation officielle Raspberry Pi. Disponible sur : <https://www.raspberrypi.com/documentation/>. Consulté le 12 mai 2026.
- [8] MITRE ATT&CK. Disponible sur : <https://attack.mitre.org/>. Consulté le 30 mai 2026.
- [9] IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems Local and Metropolitan Area Networks–Specific Requirements–Part 11 : Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.
- [10] Kismet Wireless Documentation. Disponible sur : <https://www.kismetwireless.net/docs/>. Consulté le 16 avril 2026.

- [11] Wright, J. *Detecting Wireless LAN MAC Address Spoofing and Rogue Access Points Using Passive Techniques*. SANS Institute.
- [12] Fing documentation and product resources. Disponible sur : <https://www.fing.com/>. Consulté le 3 mai 2026.
- [13] Armis platform resources. Disponible sur : <https://www.armis.com/>. Consulté le 5 mai 2026.