

BRAHIMI Massinissa

P14

RAPPORT DE 1ère PÉRIODE EN ENTREPRISE

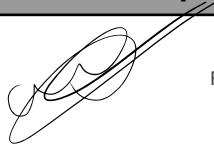
Sécurité des services et applications cloud

Dates de la période : 11//2024 – 08//2025

Entreprise / Département : Cloud Temple / Cloud Provider

Maître / Tuteur d'apprentissage : Paul LEPETIT

Tuteur pédagogique : Kamel MOULAOUI

| Visa Apprenti | Visa Entreprise |
|---|--|
|  Massinissa BRAHIMI |  Paul LEPETIT |

Remerciement

Je souhaite exprimer ma sincère gratitude à toutes les personnes qui ont contribué, directement ou indirectement, à la réussite de ma 1^{ère} année d'alternance et à l'élaboration de ce rapport.

Je tiens particulièrement à remercier mon maître d'apprentissage, Monsieur **Paul LEPETIT**, ainsi que mon tuteur académique, Kamel MOULAOUI, pour leur accompagnement rigoureux, leurs conseils éclairés et leur disponibilité tout au long de cette première année d'alternance. Leur soutien a été déterminant dans la réalisation de mes missions et dans ma progression professionnelle.

Je souhaite également remercier l'ensemble des équipes de **Cloud Temple**, et plus particulièrement l'équipe **Platforms**, pour m'avoir accueilli au sein de leurs projets, pour leur disponibilité et pour le partage généreux de leur expertise. Un remerciement tout particulier à **Yassine HADDAOUI** et à **Nicolas TORRES SANNIER** pour leur soutien constant, leur conseils pratiques, qui ont grandement facilité mon intégration et mon apprentissage.

Je dédie ce rapport à ma famille, en particulier à mes parents, pour leur confiance, leur soutien indéfectible et leurs encouragements constants tout au long de mon parcours académique.

Enfin, je tiens à remercier mes amis ainsi que le corps professoral de la **Sorbonne Université, département informatique**, et de l'**AFORP**, pour leur bienveillance, leur disponibilité et le soutien qu'ils m'ont apporté.

Nos échanges ont grandement contribué à rendre cette année d'alternance particulièrement enrichissante.

Table des matières

| | |
|---|-----------|
| Remerciement..... | 1 |
| Résumé..... | 3 |
| 1. Projets réalisés lors de mon alternance..... | 3 |
| 1.1. Sécurisation des pipelines CI/CD..... | 3 |
| 1.2. Sécurité des images Docker..... | 3 |
| 1.3. Études comparatives de solutions de sécurité..... | 4 |
| 1.4. Gestion sécurisée du partage de secrets..... | 4 |
| 2. Compétences mises en œuvre..... | 4 |
| Abstract..... | 5 |
| 1. Projects carried out during my work-study..... | 5 |
| 1.1. Securing CI/CD pipelines..... | 5 |
| 1.2. Docker image security..... | 5 |
| 1.3. Comparative studies of security solutions..... | 6 |
| 1.4. Secure secret-sharing management..... | 6 |
| 2. Skills applied..... | 6 |
| Glossaire..... | 7 |
| Liste des figures..... | 9 |
| Liste des tables..... | 10 |
| 1. Introduction..... | 11 |
| 2. Présentation de l'entreprise..... | 12 |
| 3. Travaux réalisées en entreprise..... | 13 |
| I. Intégration d'OWASP ZAP pour l'automatisation des tests DAST..... | 13 |
| A. Contexte..... | 13 |
| B. Objectif..... | 13 |
| C. Actions menées..... | 14 |
| D. Apports..... | 24 |
| II. Sécurisation des images Docker avec Trivy et Cosign..... | 24 |
| A. Contexte..... | 24 |
| C. Actions menées..... | 25 |
| D. Apports..... | 32 |
| III. Études comparatives des solutions..... | 32 |
| A. Étude comparative des WAF..... | 32 |
| B. Étude comparative des solutions de partage sécurisé de secrets..... | 36 |
| C. Apports..... | 38 |
| 4. Apports de l'expérience en entreprise..... | 39 |
| 5. Conclusion..... | 40 |
| Bibliographie..... | 41 |

Résumé

Durant mon alternance chez **Cloud Temple**, acteur majeur du cloud de confiance en France, j'ai occupé le poste de **Cloud Security Engineer Junior**.

Certifiée **SecNumCloud**, **ISO 27001** et **HDS**, l'entreprise accompagne des clients sensibles dans des secteurs critiques tels que la santé, l'industrie et la finance, en alliant sécurité, conformité et innovation dans ses services cloud.

Mon expérience s'est inscrite donc dans ce contexte hautement exigeant en matière de sécurité et de disponibilité, cela m'a permis de contribuer à des projets directement liés à la cybersécurité et à la fiabilité des infrastructures cloud.

Ma mission principale consistait à renforcer **la sécurité des services et des processus internes**. Les enjeux pour *Cloud Temple* étaient multiples :

- **Assurer la conformité réglementaire** essentielle pour conserver la confiance de ses clients issus de secteurs critiques.
- **Renforcer la posture de sécurité** de l'entreprise face à la montée des cybermenaces, les attaques ciblant les environnements cloud et les chaînes CI/CD.
- **Garantir la souveraineté numérique**, les choix technologiques doivent rester compatibles avec l'indépendance et la sécurité des données sensibles.
- **Optimiser l'efficacité opérationnelle**, en intégrant la sécurité directement dans les processus DevOps afin d'éviter des interventions correctives coûteuses en production.

1. Projets réalisés lors de mon alternance

1.1. Sécurisation des pipelines CI/CD

J'ai intégré **OWASP ZAP** dans les pipelines GitLab CI afin d'automatiser les tests de sécurité des applications internes. Cette intégration a permis de détecter des vulnérabilités dès les phases de développement, grâce à des rapports transmis aux équipes de sécurité et CSP ce qui leur a permis d'y remédier rapidement.

1.2. Sécurité des images Docker

Le second projet consistait à sécuriser les images Docker, en déployant **Trivy** pour le scan de vulnérabilités et **Cosign** pour la signature et la vérification des images, cela réduit les risques liés aux dépendances et d'assurer la traçabilité des

images, ce qui a permis de détecter et corriger les vulnérabilités avant le déploiement.

1.3. Études comparatives de solutions de sécurité

Dans notre contexte, j'ai réalisé 2 **analyses comparatives**: l'une pour sélectionner un WAF et l'autre pour identifier une solution sécurisée de partage de secrets, elles ont été présentées sous forme de tableaux et d'exposés oraux auprès de l'équipe décisionnelle, mes analyses ont été validées par mon tuteur et retenues par l'équipe décisionnelle pour orienter certains choix techniques.

1.4. Gestion sécurisée du partage de secrets

Enfin, j'ai étudié et testé des solutions telles que **Yopass** et **Cryptgeon**, destinées à sécuriser le partage secret dans le but d'éliminer les échanges en clair via des canaux non sécurisés (messageries teams, emails) et à proposer une solution interne adaptée aux besoins des équipes et des clients.

2. Compétences mises en œuvre

Ces projets m'ont permis de mobiliser un ensemble de compétences variées :

- **Techniques** : GitLab CI/CD, Python, Docker, outils de sécurité (ZAP, Trivy, Cosign) et pratiques DevSecOps.
- **Méthodologiques** : conduite d'études comparatives, rédaction de rapports techniques, synthèse des résultats sous forme de présentations.
- **Professionnelles** : travail collaboratif, communication avec des équipes pluridisciplinaires, restitution de résultats à des décideurs techniques.

En conclusion, cette alternance a constitué une expérience formatrice et enrichissante. Elle m'a permis de contribuer à l'amélioration de la sécurité des services de l'entreprise, de développer des compétences techniques solides et de renforcer ma posture professionnelle dans un environnement exigeant. Cette première année m'a donné une vision claire des enjeux de la sécurité cloud et m'a préparé à aborder ma deuxième année de master avec des bases opérationnelles robustes.

Abstract

During my work-study program at *Cloud Temple*, a leading trusted cloud provider in France, I held the position of **Junior Cloud Security Engineer**.

Certified **SecNumCloud**, **ISO 27001**, and **HDS**, the company supports sensitive clients in critical sectors such as healthcare, industry, and finance, combining security, compliance, and innovation in its cloud services.

My experience took place in this highly demanding context in terms of security and availability, allowing me to contribute to projects directly related to cybersecurity and the reliability of cloud infrastructures.

My main mission was to strengthen the security of internal services and processes. The challenges for Cloud Temple were multiple:

- Ensure regulatory compliance, essential to maintain the trust of clients from critical sectors.
- Strengthen the company's security posture against the rise of cyber threats, particularly attacks targeting cloud environments and CI/CD pipelines.
- Guarantee digital sovereignty, ensuring technological choices remain compatible with the independence and security of sensitive data.
- Optimize operational efficiency by integrating security directly into DevOps processes to avoid costly corrective actions in production.

1. Projects carried out during my work-study

1.1. Securing CI/CD pipelines

I integrated OWASP ZAP into GitLab CI pipelines to automate security testing of internal applications. This integration allowed vulnerabilities to be detected early in the development phases, with reports shared with the security and CSP teams, enabling prompt remediation.

1.2. Docker image security

The second project focused on securing Docker images by deploying Trivy for vulnerability scanning and Cosign for signing and verifying images. This reduced risks associated with dependencies and ensured image traceability, allowing vulnerabilities to be detected and corrected before deployment.

1.3. Comparative studies of security solutions

In this context, I conducted two comparative analyses: one to select a WAF and the other to identify a secure secret-sharing solution. These analyses were presented to the decision-making team in the form of tables and oral presentations, validated by my tutor, and retained by the team to guide certain technical decisions.

1.4. Secure secret-sharing management

Finally, I studied and tested solutions such as Yopass and Cryptgeon to secure secret sharing, eliminating plaintext exchanges via insecure channels (Teams, emails) and proposing an internal solution adapted to the needs of both teams and clients.

2. Skills applied

These projects allowed me to apply a range of skills:

- **Technical:** GitLab CI/CD, Python, Docker, security tools (ZAP, Trivy, Cosign), and DevSecOps practices.
- **Methodological:** conducting comparative studies, writing technical reports, and summarizing results in presentations.
- **Professional:** collaborative work, communication with multidisciplinary teams, and presenting results to technical decision-makers.

Glossaire

Cloud : Modèle d'informatique à la demande permettant d'accéder, via Internet, à des ressources hébergées dans des datacenters distants.

DevSecOps (Development, Security & Operations) : Approche intégrant la sécurité dès les premières étapes du cycle de développement logiciel

GitLab : Plateforme collaborative de gestion de code source, intégrant des fonctionnalités de versioning, de CI/CD et de gestion de projets.

Docker : Plateforme permettant de créer, déployer et exécuter des applications dans des conteneurs isolés.

ZAP Proxy : Outil de scan de sécurité open-source utilisé pour détecter les vulnérabilités dans les applications internes et publiques.

Burp Suite : Outil de test de sécurité utilisé pour l'analyse et l'audit des applications web, notamment dans la détection de failles.

Scan DAST : Méthode de test de sécurité qui analyse une application en cours d'exécution pour détecter des vulnérabilités exploitables, sans accéder au code source.

CI/CD (Continuous Integration / Continuous Deployment) : Ensemble de pratiques permettant d'automatiser l'intégration du code, l'exécution des tests et le déploiement des applications.

Template CI/CD : Fichier prédéfini utilisé pour configurer un pipeline CI/CD. Il permet d'éviter de répéter les mêmes configurations à chaque projet en automatisant des étapes comme le build, les tests et le déploiement.

API (Application Programming Interface) : Interface logicielle permettant à deux applications de communiquer entre elles.

AWS (Amazon Web Services) : Plateforme de services cloud proposée par Amazon, offrant un large éventail de solutions d'hébergement, de stockage, de calcul et de sécurité.

PoC (Proof of Concept) : Expérimentation ou prototype destiné à valider la faisabilité technique ou fonctionnelle d'une solution avant son déploiement à grande échelle.

Scheduler : Système de planification des scans de sécurité permettant de les exécuter à intervalles réguliers, sans intervention manuelle. Il couvre les URLs des applications internes et publiques définies.

MinIO : Solution de stockage d'objets open-source compatible avec l'API Amazon S3. Il permet de stocker et récupérer efficacement des fichiers et rapports générés par les scans de sécurité.

Bucket S3 : Espace de stockage cloud proposé par Amazon Web Services (AWS), utilisé ici comme dépôt centralisé pour conserver et organiser les rapports générés par les scans.

DAG (Directed Acyclic Graph) : Structure permettant de définir les dépendances entre les jobs CI/CD. Un DAG assure que les jobs sont exécutés dans le bon ordre, en respectant les prérequis et en optimisant l'exécution parallèle des tâches.

Livrables : Résultats concrets attendus dans une mission, tels que comparatifs, présentations, analyses ou démonstrations techniques.

Soft Skills : Compétences comportementales non techniques, comme la communication, le travail en équipe, la capacité de présentation et la gestion du stress.

Veille technologique : Activité consistant à surveiller et analyser l'évolution des technologies et des solutions de cybersécurité pour orienter les choix stratégiques.

Paire de clés privée/publique : Mécanisme cryptographique reposant sur deux clés distinctes et complémentaires, utilisé pour le chiffrement, la signature électronique et l'authentification.

WAF (Web Application Firewall) : Pare-feu applicatif protégeant les applications web contre les principales attaques (injections SQL, XSS, etc.).

Liste des figures

| | |
|---|----|
| Figure 01: Intégration de OWASP ZAP dans un pipeline CI/CD..... | 17 |
| Figure 02: Intégration du Scheduler dans le pipeline CI/CD..... | 18 |
| Figure 03: Exemple d'un fichier de configuration zap-config.cg..... | 19 |
| Figure 04: Fichier Docker personnalisé..... | 19 |
| Figure 05: exemple d'un job zap dans la template CI/CD..... | 20 |
| Figure 06: Création d'un DAG pour notre scan zap..... | 20 |
| Figure 07: Intégration du DAG dans le Scheduler..... | 21 |
| Figure 08: Exemple d'une planification d'un scan pour des applications ciblées..... | 22 |
| Figure 09: Exécution des tests zap sur l'application choisi..... | 23 |
| Figure 10: Résultat du scan..... | 23 |
| Figure 11: Job pour signature d'image avec cosign..... | 29 |
| Figure 12: Vérification et push vers le registre interne..... | 30 |
| Figure 13: Job de scan d'image docker avec Trivy..... | 31 |
| Figure 14: Génération des SBOM's..... | 32 |

Liste des tables

| | |
|---|----|
| Tableau 01: Comparaison des principaux outils d'analyse dynamique (DAST)..... | 15 |
| Tableau 02: Récapitulatif des résultats des scans..... | 24 |
| Tableau 03: Synthèse des solutions WAF..... | 36 |
| Tableau 04: Synthèse financière prévisionnelle (5 ans) WAF..... | 36 |
| Tableau 05: Tableau comparatif des solutions de partages de secrets..... | 38 |

1. Introduction

Ce rapport présente le bilan de mon alternance réalisée au sein de l'entreprise **Cloud Temple**, dans le cadre du **Master MSI – Parcours Sécurité des Systèmes Informatiques en apprentissage** (promotion P14). Cette formation, combinant enseignement académique et immersion en entreprise, m'a permis de consolider mes connaissances théoriques tout en développant des compétences pratiques directement applicables dans un environnement professionnel exigeant.

Dans un secteur marqué par une forte évolution des menaces et une transformation continue des infrastructures IT, l'alternance constitue une opportunité unique de confronter les acquis universitaires aux réalités opérationnelles, mes objectifs étaient avant tout de consolider mes compétences techniques, de me familiariser avec des environnements professionnels exigeants et d'apprendre à intégrer la sécurité au cœur des projets. J'ai également cherché à développer des aptitudes transverses telles que la communication, la collaboration et la gestion de projet, afin de contribuer efficacement aux missions confiées et de mieux comprendre les attentes et contraintes d'une organisation.

Ce rapport a pour objectif de présenter l'entreprise **Cloud Temple** et son environnement technologique, de retracer les principales missions auxquelles j'ai participé, et de mettre en évidence les résultats obtenus, avant de proposer une analyse des apports de cette expérience et d'ouvrir sur mes perspectives professionnelles.

Le document est structuré de la manière suivante :

- **Introduction** : présentation du contexte, de la formation et des objectifs de l'alternance.
- **Présentation de l'entreprise** : secteur d'activité, organisation, environnement technologique et rôle du service d'accueil.
- **Missions réalisées** : contexte, objectifs, actions menées et résultats obtenus.
- **Apports de l'expérience** : compétences développées et contributions à l'entreprise.
- **Conclusion et perspectives** : synthèse des résultats, enseignements tirés et ouverture sur l'avenir professionnel.

2. Présentation de l'entreprise

Cloud Temple est un acteur majeur du cloud de confiance en France, spécialisé dans l'hébergement et l'infogérance d'infrastructures critiques.

L'entreprise emploie aujourd'hui environ **233 collaborateurs** et s'adresse à des clients issus de secteurs sensibles tels que la santé, la finance, l'industrie et le secteur public. Elle est reconnue pour ses certifications de référence — **SecNumCloud [1]**, **ISO 27001 [2]** et **HDS [3]** — qui garantissent la conformité, la sécurité et la disponibilité de ses services.



L'organisation interne repose sur différents départements spécialisés. Le **département Cloud Provider** a pour mission de développer, maintenir et sécuriser les produits et services de l'entreprise.

Au sein de ce département, l'**équipe Platforms**, dirigée par **Paul Lepetit**, se concentre sur la mise en place, l'automatisation et la sécurisation des composants stratégiques des infrastructures. Cette équipe pilote des projets structurants tels que la refonte d'architectures critiques, le déploiement d'outils de sécurité (HSM, SIEM/XDR, WAF), ou encore l'automatisation de processus (Onboarding Tenant, Auto-deploy).

Dans un contexte de croissance et de modernisation continue, **Cloud Temple** investit dans des projets innovants autour de la souveraineté numérique, de l'automatisation et de la cybersécurité. Cette dynamique illustre sa volonté de renforcer sa position stratégique comme fournisseur de cloud de confiance en France.

3. Travaux réalisés en entreprise

Après une phase d'adaptation consacrée à la découverte des outils et des procédures internes de l'entreprise, j'ai bénéficié d'une courte formation qui m'a permis de me familiariser rapidement avec l'environnement technique et organisationnel de l'entreprise.

J'ai ensuite été intégré au sein de l'équipe **Platforms**, sous la responsabilité de **M. Paul Lepetit**, où j'ai progressivement commencé à contribuer aux projets en cours. Mon rôle de **Cloud Security Engineer Junior** m'a conduit à intervenir sur des problématiques concrètes liées à la sécurité des infrastructures et des services cloud.

Dans ce cadre, mes missions principales se sont articulées autour de trois axes :

- L'intégration d'**OWASP ZAP** au sein des pipelines CI/CD afin de renforcer la détection des vulnérabilités applicatives.
- Le renforcement de la **sécurité des images Docker** [8], en déployant des outils de scan et de signature.
- La réalisation d'**analyses comparatives** pour orienter le choix d'outils stratégiques.

I. Intégration d'**OWASP ZAP** pour l'automatisation des tests DAST

A. Contexte

Dans un environnement cloud soumis à de fortes exigences de sécurité et de conformité, la détection précoce des vulnérabilités applicatives constitue un enjeu majeur. Les attaques ciblant les applications web et les environnements CI/CD se multiplient, exploitant souvent des failles non corrigées dès les phases de développement. Dans ce contexte, il devenait nécessaire pour Cloud Temple d'intégrer une solution de tests de sécurité automatisés afin de réduire la surface d'attaque et d'améliorer la réactivité des équipes face aux menaces.

B. Objectif

L'objectif principal de ce projet était d'**industrialiser les tests de sécurité applicatifs** en les intégrant directement au sein des pipelines CI/CD GitLab. Il

s'agissait de permettre une détection continue des vulnérabilités, avant la mise en production, et de transmettre automatiquement les résultats aux équipes concernées (sécurité et CSP). L'enjeu était double : renforcer la sécurité « by design » et éviter les coûts liés à des correctifs tardifs en production.

C. Actions menées

Dans le cadre de ce projet, deux solutions principales ont été étudiées : **OWASP ZAP** et **Burp Suite**.

1. Tableau comparatif

| Outil | Facilité d'utilisation | Coût | Intégration CI/CD | Capacité de détection |
|------------|------------------------|--|--|--|
| OWASP ZAP | Élevée | Gratuit / open source | Intégration via CLI, API REST, plugins pour Jenkins, GitHub Actions, etc | Intégration via Burp CLI, plugins pour Jenkins, TeamCity, GitHub Actions |
| Burp Suite | Moyenne | Version Community gratuite, Pro payante (coût élevé) | Intégration via Burp CLI, plugins pour Jenkins, TeamCity, GitHub Actions | Haute précision avec peu de faux positifs, détection avancée des vulnérabilités complexes. |

Tableau 01: Comparaison des principaux outils d'analyse dynamique (DAST)

Compte tenu de ses nombreux avantages que je vais aborder dans la section suivante, le choix s'est porté sur OWASP ZAP. Cet outil répondait mieux aux contraintes du projet et permettait une adoption rapide par les équipes.

2. Présentation d'**OWASP ZAP**

OWASP ZAP [4] est un scanner de vulnérabilités open-source développé par l'OWASP, destiné à identifier les failles de sécurité dans les applications web via des

tests dynamiques (**DAST**). Contrairement aux tests statiques (**SAST**), il évalue une application en cours d'exécution, simulant ainsi des attaques réelles.

Son principal atout est sa **flexibilité** : il peut être utilisé de façon interactive (interface graphique) ou en mode automatisé via API REST et intégration dans les pipelines CI/CD.

Le choix de ZAP s'est appuyé sur plusieurs critères :

- **Open source** : Contrairement à des solutions commerciales telles que Burp Suite, Acunetix ou IBM AppScan, OWASP ZAP est entièrement gratuit et open source. Ce modèle permet à l'entreprise d'adopter un outil robuste sans coût de licence, tout en profitant du soutien d'une large communauté d'utilisateurs et de contributeurs.
- **Intégration DevSecOps** : L'outil propose une API REST complète qui facilite son intégration dans différents pipelines CI/CD (GitLab CI/CD, Jenkins, GitHub Actions, etc.). De plus, il peut être exécuté sous forme de conteneur Docker, simplifiant son déploiement et son automatisation.
- **Personnalisation avancée** : Les règles de détection des vulnérabilités peuvent être ajustées selon les besoins spécifiques grâce à des configurations fines, notamment via le fichier `zap-config.cfg`.
- **Support étendu** : Contrairement à certains outils DAST limités aux applications web traditionnelles, ZAP prend en charge les API REST et les environnements modernes basés sur les microservices. Il est également adapté aux Single Page Applications (SPA) grâce à son module AJAX Spider, capable d'explorer les interfaces dynamiques.
- **Large spectre de détection** : ZAP identifie une variété de vulnérabilités critiques, parmi lesquelles on trouve : Injection SQL, Cross-Site Scripting (XSS), Failles d'authentification et de gestion de session, Mauvaises configurations serveur, ...etc
- **Interopérabilité** : ZAP peut être combiné à d'autres outils de cybersécurité (SIEM, gestion des vulnérabilités, etc.), tels que Splunk ou Wazuh, afin de centraliser les alertes et améliorer la remédiation.
- **Maturité et adoption** : En tant que projet majeur de l'OWASP, ZAP bénéficie de mises à jour régulières, d'une documentation complète et d'un fort support

communautaire. Son adoption par de nombreuses organisations garantit également la disponibilité de scripts, plugins et bonnes pratiques.

3. Architecture de la solution

L'architecture repose sur l'orchestration de plusieurs composants interconnectés afin d'automatiser l'exécution de scans DAST et d'assurer une gestion centralisée des résultats. OWASP ZAP y est intégré pour réaliser des analyses dynamiques en simulant des attaques ciblant les applications web déployées.

- **Intégration de ZAP dans un pipeline CI/CD**

Dans un premier scénario, OWASP ZAP est intégré directement dans un pipeline CI/CD, typiquement GitLab CI. Chaque déploiement d'une application web déclenche automatiquement un scan de sécurité, permettant d'identifier d'éventuelles vulnérabilités dès les premières étapes du cycle de développement logiciel.

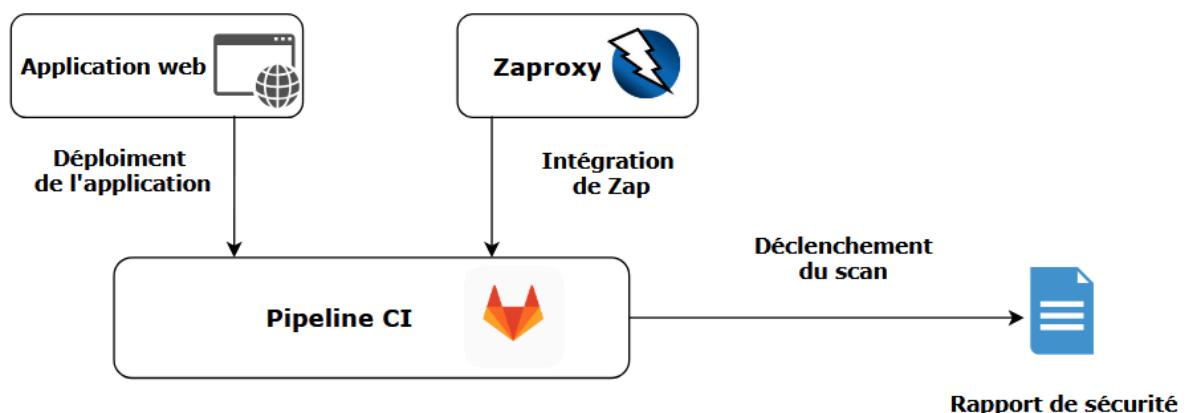


Figure 01: Intégration de OWASP ZAP dans un pipeline CI/CD

- **Mise en place d'un Scheduler et gestion centralisée des résultats**

Un second scénario repose sur l'ajout d'un scheduler au sein du pipeline CI/CD. Ce mécanisme permet de planifier l'exécution régulière des tests dynamiques réalisés par OWASP ZAP, sans intervention manuelle.

Les analyses génèrent des rapports de sécurité, stockés de manière centralisée dans un bucket S3 compatible MinIO [11]. Cette approche garantit la conservation, la traçabilité et la disponibilité des résultats. Les équipes Sécurité et

CSP disposent ainsi d'un accès simplifié aux rapports, ce qui facilite leur analyse et accélère la mise en œuvre de mesures correctives.

Ce dispositif assure une surveillance continue, automatisée et fiable de la surface d'attaque applicative.

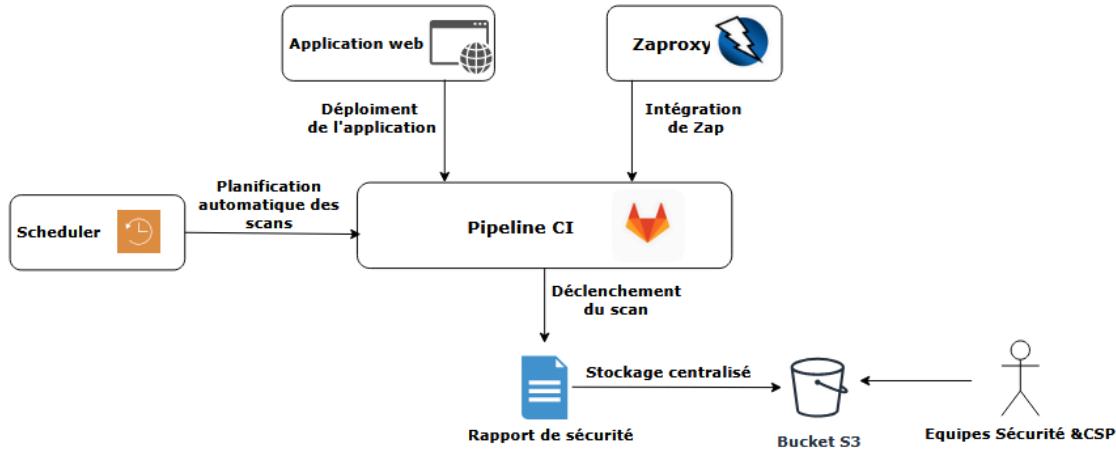


Figure 02: Intégration du Scheduler dans le pipeline CI/CD

4. Mise en œuvre de la solution

- **Création de l'image Docker ZAP personnalisée**

Une image Docker personnalisée, construite à partir de la version stable d'OWASP ZAP (zap-stable:2.15.0 au moment de la réalisation), a été développée afin de répondre aux besoins spécifiques du projet. Cette image intègre directement un client **MinIO**, permettant le transfert automatisé des rapports générés vers un stockage objet compatible S3.

Afin d'affiner le processus de détection, un fichier de configuration dédié (**zap-config.cfg**) a été ajouté. Celui-ci permet de définir précisément les règles de scan à appliquer ainsi que leur criticité, en fonction des priorités de l'analyse de sécurité. Chaque règle peut ainsi être associée à un niveau de严重性 spécifique :

- **WARN** : génère un avertissement sans interrompre le scan,
- **IGNORE** : exclut la règle afin de réduire le bruit lié aux faux positifs,
- **FAIL** : bloque immédiatement le scan lorsqu'une vulnérabilité critique est détectée.

The screenshot shows a GitHub commit titled "add conf file to our base image" by Massinissa BRAHIMI. The file "zap-config.cfg" contains the following configuration:

```

1 # zap-api-scan rule configuration file
2 # Change WARN to IGNORE to ignore rule or FAIL to fail if rule matches
3 # Active scan rules set to IGNORE will not be run which will speed up the scan
4 # Only the rule identifiers are used - the names are just for info
5 # You can add your own messages to each rule by appending them after a tab on each line.
6 0    WARN   (Directory Browsing - Active/release)
7 10010  WARN  (Cookie No HttpOnly Flag - Passive/release)
8 10011  WARN  (Cookie Without Secure Flag - Passive/release)
9 10012  WARN  (Password Autocomplete in Browser - Passive/release)
10 10015  WARN  (Incomplete or No Cache-control and Pragma HTTP Header Set - Passive/release)
11 10016  WARN  (Web Browser XSS Protection Not Enabled - Passive/release)
12 10017  WARN  (Cross-Domain JavaScript Source File Inclusion - Passive/release)
13 10019  WARN  (Content-Type Header Missing - Passive/release)
14 10020  WARN  (X-Frame-Options Header Scanner - Passive/release)
15 10021  WARN  (X-Content-Type-Options Header Missing - Passive/release)
16 10023  WARN  (Information Disclosure - Debug Error Messages - Passive/beta)
17 10024  WARN  (Information Disclosure - Sensitive Informations in URL - Passive/beta)

```

Figure 03: Exemple d'un fichier de configuration zap-config.cfg

le fichier Dockerfile utilisé pour la création de l'image personnalisée

The screenshot shows a GitHub commit titled "add conf file to our base image" by Massinissa BRAHIMI. The Dockerfile contains the following instructions:

```

1 FROM registry.cti.lan/hub/zaproxy/zap-stable:2.15.0
2 USER root
3 COPY certificates/* /usr/local/share/ca-certificates/
4 RUN apt-get install ca-certificates && update-ca-certificates
5 RUN apt-get update && \
6     apt-get install -y python3 python3-pip
7
8 RUN curl -O https://dl.min.io/client/mc/release/linux-amd64/mc && \
9     chmod +x mc && \
10    mv mc /usr/local/bin/
11
12 COPY zap-config.cfg /zap
13
14 USER zap
15

```

Figure 04: Fichier Docker personnalisé

Une fois l'image construite, celle-ci a été poussée vers le registre interne de l'entreprise (*registry.cti.lan*).

- **Création d'un template CI/CD**

Un **job ZAP** a été défini au sein d'un template CI/CD générique, réutilisable pour l'ensemble des projets. Ce template regroupe l'ensemble des étapes nécessaires : lancement de l'instance ZAP, application de la configuration, exécution du scan de sécurité, puis transfert automatique du rapport généré vers le stockage S3.

```

40 # Script d'automatisation du scan ZAP
41 # Ce script effectue un scan de sécurité sur une liste d'applications, sites cibles, génère des rapports,
42 # et les transfère vers un bucket S3 (minIO).
43 # Il inclut une gestion des noms de fichiers pour éviter les caractères spéciaux.
44 .zap_scan: &zap_scan
45   - echo "[*] Running ZAP baseline scan..."
46   - mkdir -p /zap/wrk/
47   - |
48     mc alias set 'vidhar' "${S3_BASE_URL}" "${BUCKET_SECRET_ACCESS_KEY}" "${BUCKET_SECRET_SECRET_KEY}"
49     ORIGINAL_IFS="$IFS"
50     IFS=","
51     read -ra TARGETS_ARRAY <<< "$TARGETS"
52     IFS="$ORIGINAL_IFS"
53
54     COUNTER=1
55     for TARGET in "${TARGETS_ARRAY[@]}"; do
56       echo "[*] Scanning: $TARGET"
57       Name_TARGET=$(echo "$TARGET" | sed -E "s|https://||g" | sed "s|[:.]|_|g" | sed "s|_|_|g" | sed "s|_|$|g")
58       REPORT_FILE="zap_scan_${Name_TARGET}.html"
59       /zap/zap-baseline.py -t "$TARGET" -r "$REPORT_FILE" -I -c /zap/zap-config.cfg
60       echo "[*] Scan completed for $TARGET"
61       echo "[*] Transferring the report ${REPORT_FILE} for $TARGET"
62       mc cp /zap/wrk/"${REPORT_FILE}" "vidhar/${BUCKET_NAME}/"
63       ((COUNTER++))
64     done
65
66   - echo "[+] All scans and reports have been completed"

```

Figure 05: exemple d'un job zap dans la template CI/CD

- Mise en place d'un DAG dans le pipeline**

L'intégration d'un **DAG (Directed Acyclic Graph)** au sein du pipeline CI/CD permet d'orchestrer l'exécution des différentes tâches de manière optimale, tout en respectant les dépendances entre elles.

The screenshot shows a GitHub commit interface. At the top, it says "changement du nom de DAG" and "Massinissa BRAHIMI authored 5 months ago". On the right, there are buttons for "Edit", "Replace", "Delete", and "History". Below this, the file content is displayed:

```

zap_scan.yml 281B
Edit Replace Delete History

1 scan_with_zap:
2   extends: .scheduler
3   variables:
4     SCHEDULER_IMAGE: ${SCHEDULER_IMAGE_ZAP}
5     TARGETS: ""
6     BUCKET_NAME: ""
7     S3_BASE_URL: ""
8     BUCKET_SECRET_ACCESS_KEY: ""
9     BUCKET_SECRET_SECRET_KEY : ""
10    before_script: []
11    script:
12      - !reference [.zap_scan]

```

Figure 06: Création d'un DAG pour notre scan zap

Intégration du DAG dans le Pipeline :



The screenshot shows a GitLab pipeline interface. At the top, there's a header with a purple circular icon, the text "changement de branche", and "Massinissa BRAHIMI authored 5 months ago". To the right are buttons for "Edit", "Replace", "Delete", and "History". Below this, the pipeline status shows a green checkmark and the commit hash "7b99a825". The main area displays the ".gitlab-ci.yml" file content:

```
1 include:
2   - project: 'tools/gitlab-ci-templates'
3     ref: zap-schedule-targets
4     file:
5       - /scheduler.yml
6   - local: dags/${DAG_NAME}.yml
7   rules:
8     - exists:
9       - dags/${DAG_NAME}.yml
```

Figure 07: Intégration du DAG dans le Scheduler

● Planification des scans

Le scan est configuré pour s'exécuter automatiquement à intervalles réguliers (toutes les X minutes par exemple) sur les applications ciblées.

Cela assure une surveillance continue et permet de détecter rapidement de nouvelles vulnérabilités. Les résultats étant historisés, il est également possible de suivre l'évolution de la sécurité dans le temps et de vérifier l'efficacité des correctifs appliqués.

Edit Scheduled Pipeline

Description

zap scan

Cron timezone

[UTC+2] Paris

Interval Pattern

- Every day (at 3:20pm)
- Every week (Monday at 3:20pm)
- Every month (Day 11 at 3:20pm)
- Custom

*/2 * * * *

Set a custom interval with Cron syntax. [What is Cron syntax?](#)**Select target branch or tag**

zap-scheduler

Variables

| | | | |
|----------|--------------------|----------------------|--|
| Variable | DAG_NAME | ***** | |
| Variable | TARGETS | ***** | |
| Variable | Input variable key | Input variable value | |

Reveal values Activated

Figure 08: Exemple d'une planification d'un scan pour des applications ciblées

- **Analyse des résultats**

Exemple d'application scannée: <https://shiva-api.domain.local/>

```

183 [*] Scanning: https://api.shiva.rec.lan/
184 Using the Automation Framework
185 Total of 4 URLs
186 PASS: Vulnerable JS Library (Powered by Retire.js) [10003]
187 PASS: Cookie No HttpOnly Flag [10010]
188 PASS: Cookie Without Secure Flag [10011]
189 PASS: Re-examine Cache-control Directives [10015]
190 PASS: Cross-Domain JavaScript Source File Inclusion [10017]
191 PASS: Content-Type Header Missing [10019]
192 PASS: Anti-clickjacking Header [10020]
193 PASS: X-Content-Type-Options Header Missing [10021]
194 PASS: Information Disclosure - Debug Error Messages [10023]

```

Figure 09: Exécution des tests zap sur l'application choisi

```

249 [+] Scan completed for https://api.shiva.rec.lan/
250 [*] Transferring the report zap_scan_api_shiva_rec_lan.html for https://api.shiva.rec.lan/
251 `zap/wrk/zap_scan_api_shiva_rec_lan.html` -> `vidhar/scan-dast/zap_scan_api_shiva_rec_lan.html`

252
253 ┌─────────┐ ┌─────────┐ ┌─────────┐ ┌─────────┐
254 ┌── Total ──┐ ┌── Transferred ──┐ ┌── Duration ──┐ ┌── Speed ──┐
255 └── 20.67 KiB ──┘ └── 20.67 KiB ──┘ └── 00m00s ──┘ └── 431.44 KiB/s ──┘
256
257 $ echo "[+] All scans and reports have been completed"
258 [+] All scans and reports have been completed
259 Cleaning up project directory and file based variables
260 Updating CA certificates...
261 WARNING: ca-cert-ca.pem does not contain exactly one certificate or CRL: skipping
262 Job succeeded

```

Figure 10: Résultat du scan

Le scan OWASP ZAP s'est achevé avec succès en identifiant les vulnérabilités présentes sur les différentes URL analysées. Les rapports générés ont été automatiquement centralisés dans le bucket S3, offrant aux équipes de sécurité un accès direct pour analyser les résultats et engager les actions correctives nécessaires.

• Classification des vulnérabilités

Les vulnérabilités détectées sont classées par niveaux de criticité afin de faciliter leur priorisation et leur traitement :

- **Critique** : vulnérabilités à fort impact nécessitant une correction immédiate.
- **Moyenne** : failles importantes devant être corrigées rapidement.
- **Faible** : vulnérabilités mineures, avec un risque limité.

- **Informationnel** : éléments à caractère informatif, ne représentant pas une menace directe.
- **Faux positifs** : résultats erronés identifiés et exclus du périmètre de correction.

Afin de valider notre solution, des tests ont été réalisés sur plusieurs applications en environnement de recette. Le tableau ci-dessous illustre un exemple de résultats obtenus suite à l'exécution de nos scans :

| Application | Critique | Moyenne | Faible | Info | Faux Positif | Total |
|-------------|----------|---------|--------|------|--------------|-------|
| True API | 0 | 0 | 1 | 0 | 0 | 1 |
| Saturne NG | 0 | 0 | 8 | 0 | 0 | 8 |
| API Shiva | 0 | 0 | 1 | 0 | 0 | 1 |

Tableau 02: Récapitulatif des résultats des scans

Les résultats obtenus montrent exclusivement des vulnérabilités de **niveau faible**, sans détection d'anomalies critiques ou moyennes.

Bien que leur impact soit limité, ces alertes doivent être traitées par les équipes concernées afin de réduire la surface d'attaque.

Le rapport généré par ZAP fournit non seulement une **analyse détaillée** des vulnérabilités, mais également des **recommandations de correction** adaptées. Ces éléments permettent aux équipes de sécurité d'anticiper les risques et de mettre en œuvre les correctifs nécessaires avant tout passage en production.

Cependant, certaines limites ont été identifiées : il est nécessaire d'affiner les règles de détection pour réduire les faux positifs et optimiser la performance des scans.

Par ailleurs, bien qu'OWASP ZAP soit efficace pour le DAST, il ne couvre qu'une partie des tests de sécurité et doit être complété par d'autres approches, telles que les tests SAST et SCA, afin d'assurer une couverture de sécurité plus complète.

D. Apports

Pour l'entreprise:

- Mise en place d'une démarche DevSecOps, en intégrant la sécurité dès les phases de développement,
- Réduction des risques liés aux vulnérabilités applicatives grâce à une détection précoce,
- Gain en efficacité opérationnelle en limitant les corrections post-déploiement.

Pour moi :

- Acquisition d'une compréhension approfondie des tests dynamiques, des vulnérabilités courantes et de leur impact sur les applications web.
- Expérience pratique dans l'intégration d'outils de sécurité au sein de pipelines CI/CD.
- montée en compétence sur la configuration avancée d'OWASP ZAP et son orchestration dans un pipeline automatisé.
- prise de recul sur les enjeux stratégiques et opérationnels liés à la sécurisation des applications dans un environnement cloud réglementé.

II. Sécurisation des images Docker avec Trivy et Cosign

A. Contexte

Dans un environnement cloud fortement basé sur les conteneurs, la sécurité des images Docker est un enjeu critique.

Historiquement, les entreprises faisaient confiance aux images téléchargées depuis Docker Hub ou d'autres registres publics, mais plusieurs risques majeurs sont apparus :

- **Images malveillantes** : certaines intègrent des backdoors, malwares ou scripts de crypto-jacking.
- **Manque de traçabilité** : sans signature, impossible de garantir qui a construit l'image et si elle a été modifiée après sa création.
- **Usurpation d'identité** : des attaquants publient des images ressemblant à des versions officielles (ex. `nginx-officiel`) pour tromper les utilisateurs.

Avec la généralisation de Kubernetes et du CI/CD, ces vulnérabilités se propagent encore plus vite :

- une **image compromise** peut être automatiquement déployée et contaminer l'ensemble de l'infrastructure,
- les **attaques supply chain** (comme SolarWinds) ont montré la gravité d'une compromission à la source.

B. Objectif

L'objectif principal de cette mission était double :

- **Détection des vulnérabilités:** Scanner automatiquement les images Docker avec **Trivy** pour identifier les failles de sécurité présentes dans les dépendances ou packages systèmes.
- **Signature et validation des images:** Mettre en place la **signature cryptographique des images** avec **Cosign** afin de garantir :
 - leur **intégrité** (elles n'ont pas été modifiées après création),
 - leur **authenticité** (elles proviennent bien d'une source légitime).
- **Sécurisation des pipelines CI/CD:** Intégrer ces mécanismes dans les pipelines GitLab CI/CD afin de n'autoriser le déploiement que d'**images signées et validées**, réduisant ainsi les risques de supply chain.

C. Actions menées

Afin de renforcer la confiance et la traçabilité des images utilisées dans les pipelines, deux mécanismes complémentaires ont été mis en place :

- **Cosign** [10], pour assurer la **signature cryptographique** des images et garantir leur intégrité et leur authenticité,
- **Trivy** [9], pour effectuer une **analyse de vulnérabilités** systématique et produire un inventaire complet des dépendances (SBOM).

Dans un premier temps, j'ai souhaité me familiariser avec le concept de signature et de vérification d'images Docker avant d'intégrer des solutions plus avancées comme **Cosign** dans le pipeline CI/CD.

Pour cela, j'ai expérimenté localement la fonctionnalité **Docker Content Trust (DCT)**, proposée nativement par Docker,

Cette phase exploratoire m'a permis de :

- **Activer et tester DCT** sur ma machine afin de contraindre Docker à n'accepter que des images signées.
- **Vérifier les digests SHA-256 des images** téléchargées, garantissant que le conteneur lancé correspond exactement à l'image attendue.
- **Inspecter les signatures d'images existantes** (par exemple `alpine:latest`) afin de comprendre la relation entre **tags, digests et signatures**.
- **Analyser les métadonnées de confiance** (administrative keys, repository keys, signers) pour identifier le rôle des différentes clés cryptographiques dans le processus de validation.

Cette première étape m'a permis de consolider ma compréhension des mécanismes fondamentaux de la signature, notamment :

- La signature ne concerne pas directement une image brute, mais **l'association entre un tag et un digest spécifique**, garantissant la traçabilité et l'authenticité.
- La distinction entre les rôles des **clés Root et Repository**, qui illustre la hiérarchie des clés de confiance.

À l'issue de mes tests, j'ai cependant constaté plusieurs limites rendant **DCT peu adapté aux environnements d'entreprise** :

- **Pas de support des signatures multiples** : impossible pour plusieurs équipes (dev, sécurité, conformité) de signer une même image.
- **Dépendance forte à Docker Hub** : DCT est lié à Notary v1 et fonctionne principalement avec Docker Hub, limitant son utilisation avec des registres modernes (ECR, GCR, GHCR...).

- **Gestion rigide des clés** : les clés sont stockées localement, sans intégration avec des solutions sécurisées comme HSM ou KMS.
- **Absence d'auditabilité avancée** : pas de traçabilité fine sur *qui* a signé une image et *quand*.
- **Obsolescence de Notary v1** : Docker a déjà annoncé la transition vers Notary v2, ce qui rend DCT progressivement dépassé.

Ces limites m'ont poussé à explorer des solutions plus modernes et adaptées, comme Cosign (reposant sur Sigstore), et par la même occasion cela m'a permis de comprendre pourquoi mon équipe a choisi cette solution.

1. Mise en place de Cosign:

Après avoir testé Docker Content Trust (DCT) et identifié ses limitations, j'ai commencé à travailler sur **Cosign**, une solution moderne pour sécuriser nos images Docker. L'objectif était de garantir **l'authenticité et l'intégrité des images** tout en s'intégrant facilement à notre pipeline CI/CD et à nos registres privés.

- **Préparation et variables**

La première étape a été de définir les **variables globales** nécessaires pour Cosign, Vault et Harbor [12]. Cela inclut la version de Cosign, les chemins des clés, les identifiants Vault pour la gestion sécurisée des secrets, ainsi que les informations pour accéder au registre Harbor. Ces variables assurent que le pipeline est **automatisable, reproductible et sécurisé**.

- **Gestion des clés Cosign**

Pour signer et vérifier les images, il faut une **paire de clés privée/publique**. Le pipeline gère les clés de manière sécurisée en :

1. Vérifiant si elles existent déjà dans Vault.
2. Générant de nouvelles clés si nécessaire, avec un mot de passe aléatoire pour sécuriser la clé privée.
3. Récupérant les clés depuis Vault pour les jobs de la CI avec des permissions sécurisées.

- **Signature des images**

La signature se fait pour toutes les images utilisées dans le pipeline, notamment GitLab et Harbor. Le processus comprend :

1. **Pull des images à signer**, en utilisant le SHA pour éviter toute ambiguïté liée aux tags.
2. **Construction du chemin Harbor** en fonction du projet et du type d'image (applications, tools, projets spécifiques).
3. **Signature avec Cosign**, en utilisant les clés stockées dans Vault et en protégeant le mot de passe.

```

217 # Template pour la signature d'une image avec Cosign
218 .cosign_sign: &cosign_sign
219   # HARBOR_IMAGE="${HARBOR_REGISTRY}/${HARBOR_PROJECT}/${CI_PROJECT_NAME}"
220   -
221   echo "Signature de l'image avec Cosign..."
222   # Créer un fichier temporaire pour le mot de passe Cosign
223   COSIGN_PWD_FILE=$(mktemp)
224   echo "${COSIGN_PASSWORD}" > ${COSIGN_PWD_FILE}
225
226   # Pull image Gitlab
227   echo "Pull image Gitlab"
228   docker pull "${DOCKER_IMAGE}:${DOCKER_TAG}"
229   # Obtenir le SHA de l'image Gitlab pour la signer par SHA plutôt que par tag
230   IMAGE_SHA=$(docker inspect --format='{{index .RepoDigests 0}}' ${DOCKER_IMAGE}:${DOCKER_TAG})
231   echo "image sha"
232   echo ${IMAGE_SHA}
233   - !reference [.harbor_image_path]
234   -
235   # Pull image Harbor
236   echo "Pull image Harbor"
237   docker pull "${HARBOR_IMAGE}:${HARBOR_TAG}"
238   # Obtenir le SHA de l'image Harbor pour la signer par SHA plutôt que par tag
239   HARBOR_IMAGE_SHA=$(docker inspect --format='{{index .RepoDigests 0}}' ${HARBOR_IMAGE}:${DOCKER_TAG})
240   echo "harbor image sha"
241   echo ${HARBOR_IMAGE_SHA}
242
243   # Signature des images Gitlab et Harbor
244   echo "Signing gitlab image par SHA: ${IMAGE_SHA}..."
245   COSIGN_PASSWORD_STDIN=true cat ${COSIGN_PWD_FILE} | cosign sign --tlog-upload=false --key ${COSIGN_KEY_PATH} --yes ${IMAGE_SHA}
246
247   echo "Signing harbor image par SHA: ${HARBOR_IMAGE_SHA}..."
248   COSIGN_PASSWORD_STDIN=true cat ${COSIGN_PWD_FILE} | cosign sign --tlog-upload=false --key ${COSIGN_KEY_PATH} --yes ${HARBOR_IMAGE_SHA}
249
250   # Supprimer le fichier temporaire
251   rm -f ${COSIGN_PWD_FILE}

```

Figure 11: Job pour signature d'image avec cosign

- **Vérification et push**

- **Vérification** : les signatures des images sont validées avec les clés publiques, en mode offline pour ne pas dépendre d'un serveur externe.
- **Push** : seules les images signées et vérifiées sont taggées et poussées vers Harbor, assurant que l'environnement de production reçoit uniquement des images fiables.

```

254 # Template pour la vérification de la signature d'une image avec Cosign
255 .cosign_verify: &cosign_verify
256     # HARBOR_IMAGE="${HARBOR_REGISTRY}/${HARBOR_PROJECT}/${CI_PROJECT_NAME}"
257     - |
258         export HTTP_PROXY=${PROXY_URL}
259         export HTTPS_PROXY=${PROXY_URL}
260         export NO_PROXY=${UNPROXIED}
261     - !reference [.harbor_image_path]
262     - |
263         # Vérifier la signature de l'image
264         echo "Vérification de la signature de l'image Gitlab avec Cosign..."
265         cosign verify --key /tmp/cosign.pub --insecure-ignore-sct=true --insecure-ignore-tlog=true --offline ${DOCKER_IMAGE}:${DOCKER_TAG}
266         echo "Vérification de la signature de l'image Harbor avec Cosign..."
267         echo ${DOCKER_TAG}
268         echo ${HARBOR_IMAGE}
269         cosign verify --key /tmp/cosign.pub --insecure-ignore-sct=true --insecure-ignore-tlog=true --offline ${HARBOR_IMAGE}:${DOCKER_TAG}
270
271 ######
272 # TEMPLATES HARBOR #
273 #####
274 # Template pour le push vers Harbor
275 .harbor_push: &harbor_push
276     # HARBOR_IMAGE="${HARBOR_REGISTRY}/${HARBOR_PROJECT}/${CI_PROJECT_NAME}"
277     - !reference [.harbor_image_path]
278     - |
279         echo "Push de l'image vers Harbor..."
280         # Tagger l'image pour Harbor
281         docker tag "${DOCKER_IMAGE}":${DOCKER_TAG} "${HARBOR_IMAGE}":${DOCKER_TAG}"
282         # Push l'image vers Harbor
283         docker push "${HARBOR_IMAGE}":${DOCKER_TAG}"

```

Figure 12: Vérification et push vers le registre interne

2. Analyse avec Trivy :

En complément du mécanisme de signature, un processus d'**analyse de vulnérabilités** a été intégré aux pipelines à l'aide de **Trivy**. L'objectif est de garantir que les images utilisées en production ne présentent pas de failles critiques connues.

Avant toute mise en production, chaque image suit un processus structuré :

1. Elle est scannée pour identifier d'éventuelles vulnérabilités critiques,
2. Elle est documentée par un **SBOM** généré automatiquement,
3. Les résultats sont centralisés dans un espace de stockage accessible,
4. Enfin, seules les images analysées, tracées et signées (via Cosign) peuvent atteindre les environnements de production.

Le processus repose ensuite sur plusieurs étapes successives :

- **Construction et scan d'image :**

L'image Docker est d'abord construite en intégrant des métadonnées (nom du projet, date, hash Git). Cela permet d'assurer une traçabilité complète.

L'image est scannée par Trivy, avec un paramétrage qui cible les vulnérabilités de niveau **élevé et critique**. Les résultats sont générés sous forme de rapport HTML grâce à un template dédié.

```

67
68     echo "[+] analyse de l'image avec Trivy..."
69
70     # mode de MEP par défaut (scan + push après)
71     if [ "$ALLOW_HARBOR_UPLOAD" = "false" ]; then
72         echo "[+] mode de MEP par défaut"
73         ./trivy --cache-dir .trivycache/ image \
74             --exit-code 0 \
75             --severity "$TRIVY_SEVERITY" \
76             --no-progress \
77             --format template \
78             --template "@html.tpl" \
79             -o trivy-report.html \
80             "$TRIVY_IMAGE"
81         docker push "$TRIVY_IMAGE"
82     else
83         # mode MEP immédiat (push immédiat et scan en parallèle)
84         echo "[+] mode MEP immédiat "
85         docker push "$TRIVY_IMAGE"
86         ./trivy --cache-dir .trivycache/ image \
87             --exit-code 0 \
88             --severity "$TRIVY_SEVERITY" \
89             --no-progress \
90             --format template \
91             --template "@html.tpl" \
92             -o trivy-report.html \
93             "$TRIVY_IMAGE"
94     fi
95     - export REPORT_FILE=trivy-report.html
96     - *report_upload

```

Figure 13: Job de scan d'image docker avec Trivy

- **Modes de déploiement :**

- **Mode par défaut** : l'image est scannée avant d'être poussée vers Harbor.
- **Mode immédiat** : l'image est poussée rapidement vers Harbor, le scan étant effectué en parallèle.
Cette logique permet de concilier rapidité de livraison et sécurité, selon le besoin opérationnel.

- **Stockage et accessibilité des rapports :**

Une fois généré, le rapport est automatiquement envoyé dans un **bucket S3** dédié. Cela garantit une conservation centralisée et un accès facilité pour les équipes de développement et de sécurité.

- **Génération d'un SBOM**

En complément du scan, un **SBOM (Software Bill of Materials)** est produit à l'aide de l'outil **Syft**. Ce fichier, exporté au format **SPDX**, dresse la liste exhaustive des dépendances utilisées par l'image.

Ce document est essentiel pour :

- Suivre précisément les composants tiers intégrés,
- Identifier rapidement les dépendances vulnérables,
- Répondre aux exigences de conformité (par exemple NIST, ISO 27001).

```
102 .generate_sbom:
103   stage: sboms
104   image: registry.cti.lan/internal/ci-base/docker:v1.0.14-rc.2
105
106   before_script:
107     - apk add --no-cache syft
108     - *docker_login
109     - *mc_client_dl
110
111   script:
112     - syft "$TRIVY_IMAGE" -o spdx-json > sbom.json
113     - export REPORT_FILE=sbom.json
```

Figure 14: Génération des SBOM's

La logique suivie est la suivante :

- **Avant mise en production**, chaque image est **scannée** pour identifier les vulnérabilités critiques et documentées par un SBOM.
- **Les résultats** sont stockés dans un espace centralisé, permettant une exploitation transversale par les différentes équipes.
- **La pipeline garantit** ainsi que seules des images analysées, tracées et signées (via Cosign) peuvent atteindre les environnements de production.

D. Apports

Pour l'entreprise:

- Mise en place d'une chaîne de confiance autour des images Docker, garantissant leur authenticité et leur intégrité.
- Réduction du risque d'introduction d'images compromises dans les environnements de production.
- Alignement avec les normes et bonnes pratiques de sécurité (NIST, ISO 27001, etc.), renforçant la conformité des processus internes.
- Amélioration de la résilience face aux attaques de type supply chain, grâce à une meilleure traçabilité et un contrôle accru des artefacts.

Pour moi:

- Découverte et mise en œuvre des outils Trivy et Cosign, outils que je n'avais pas utilisé auparavant

III. Études comparatives des solutions

Dans le cadre du projet, deux études comparatives ont été réalisées afin de guider le choix de solutions techniques adaptées à notre environnement interne :

1. **Sélection d'un WAF (Web Application Firewall)**
2. **Sélection d'une solution de partage sécurisé de secrets**

Chaque étude s'est appuyée sur une méthodologie structurée : définition des critères, analyse des solutions candidates, réalisation de tests ou PoC(cas pour la 2^{ème} étude), puis production de livrables synthétiques pour faire une présentation à l'équipe.

A. Étude comparative des WAF

1. Objectifs et méthodologie

L'objectif de cette étude était de sélectionner un **Web Application Firewall (WAF)** adapté à notre infrastructure interne.

La méthodologie a consisté à définir un ensemble de **critères de sélection**

structurés en catégories, puis à évaluer les solutions candidates selon ces critères.

Les catégories de critères retenues sont :

- **Souveraineté:** Respect des réglementations européennes / solution open-source
- **Finance:** Coût de licence , coût humain, coût cyber, coût matériel (HW), opportunités commerciales (WAF SNC)
- **MCO/MCS (Maintien en Condition Opérationnelle / Sécurité):** Support de la solution, popularité, pérennité de l'éditeur, facilité d'intégration, facilité d'administration et d'automatisation, support disponible (24/7), sécurité logicielle
- **Fonctionnel:** Capacité de détection et de réponse (D&R), facilité d'utilisation
- **Technique:** Haute disponibilité (HA) / mécanismes de backup, Journalisation et traçabilité, intégration SSO ou LDAPS, type de tenant (multi-tenant par organisation ou par IP)

Ces critères ont permis d'assurer une **évaluation objective et complète**, en tenant compte non seulement des aspects techniques et sécuritaires, mais aussi des dimensions financières, opérationnelles et stratégiques.

Trois solutions ont été étudiées :

- **Imperva WAF (Thales, On-Premise)[5]**
- **HAProxy Enterprise Edition (On-Premise)**
- **BunkerWeb (On-Premise, Open Source)**

L'étude a reposé sur plusieurs étapes complémentaires :

- **Recherche documentaire approfondie** menée sur chaque solution, en s'appuyant sur les sources officielles (sites éditeurs, dépôts GitHub, forums techniques, rapports Gartner, etc.).
- **Évaluation structurée** des solutions selon des critères fonctionnels, techniques, financiers et opérationnels, afin d'assurer une analyse exhaustive

et objective.

- **Production de livrables** destinés à l'équipe projet incluant une documentation générale, un tableau comparatif détaillé (Excel) et une présentation synthétique (PowerPoint) facilitant la prise de décision collective.

2. Synthèse des solutions

| #1 <u>Imperva WAF</u> On-Premise, by Thales | #2 <u>HA Proxy</u> On-Premise, Enterprise Edition (Security Solutions) | #3 <u>BunkerWeb</u> On-Premise, Enterprise Edition |
|---|--|--|
| <p>+++++ Confiance : Solution leader, image forte et justifié auprès des RSSI/DT</p> <p>+++++ Sécurité : Capacité de détection et réponse nettement supérieur, avec un taux de faux positif très bas (impact sécurité et financier), embarquant de nombreuses fonctionnalités de sécurité</p> <p>+++++ Durabilité : Solution très pérenne, leader mondial en cybersécurité, top 1 Gartner</p> <p>+++++ Facilité : De déploiement, d'administration et de mise en place de règles de détection qualitative</p> | <p>++/+++ Confiance : Entreprise sérieuse et relativement innovante, mais peu de retour public sur la solution WAF</p> <p>++/+++ Sécurité : Capacité de détection et réponse basé sur ModSecurity, avec des fonctionnalités en plus (anti DDOS, anti BOT, IA)</p> <p>+++/++++ Durabilité : Entreprise connue pour sa fiabilité au niveau HA Proxy LB, 130 employés</p> <p>++/+++ Facilité : Le déploiement et l'administration est simple mais la partie D&R risque d'être plus complexe</p> | <p>++/+++ Confiance : Solution WAF qui commence à être populaire dans le monde de l'open source depuis 2021</p> <p>++ Sécurité : Capacité de détection et réponse basé sur ModSecurity, avec des fonctionnalités en plus (anti DDOS, anti BOT)</p> <p>++ Durabilité : Entreprise connue dans le monde de l'open source, mais un seul développeur et résultat financier privé</p> <p>++ Facilité : Le déploiement et l'administration est simple mais la partie D&R est plus complexe</p> |

| | | |
|---|--|--|
| <p>-/- Finance : Coût de licence probablement supérieur -> attente de devis.</p> <p>MAIS, à déduire de la diminution des coûts humains (rapidité d'intégration des règles, taux de faux positif extrêmement faible), des risques cyber maîtrisés, de la durabilité et des opportunités commerciales futurs</p> | <p>-- Confiance : Très peu de retours sur la solution WAF -> attente Q/R</p> <p>-- Sécurité : Très probablement plus faible que la solution Imperva -> attente Q/R</p> <p>-- Finance : Coût humain et cyber plus important</p> | <p>-- Confiance : Risque durabilité, innovation et support</p> <p>-- Sécurité : Plus faible que la solution Imperva</p> <p>--/-- Finance : Coût humain et cyber plus important</p> |
|---|--|--|

Tableau 03: Synthèse des solutions WAF

3. Synthèse financière prévisionnelle (5 ans)

Une analyse financière prévisionnelle a été réalisée sur un horizon de 5 ans (WIP – Work in Progress) pour comparer les trois solutions étudiées :

| #1 <u>Imperva WAF</u> On-Premise, by Thales | #2 <u>HA Proxy</u> On-Premise, Enterprise Edition (Security Solutions) | #3 <u>BunkerWeb</u> On-Premise, Enterprise Edition |
|--|--|--|
| <ul style="list-style-type: none"> Coût de license : 5-6 Coût humain : 8 Coût technique : 5-7 Coût cyber : 10 Business impact : 8-10 | <ul style="list-style-type: none"> Coût de license : 6-8 Coût humain : 5-6 Coût technique : 7-8 Coût cyber : 5-6 Business impact : 6-7 | <ul style="list-style-type: none"> Coût de license : 8-9 Coût humain : 5-6 Coût technique : 7-8 Coût cyber : 5-6 Business impact : 5-6 |

Tableau 04: Synthèse financière prévisionnelle (5 ans) WAF

4. Analyse:

- Imperva WAF** reste la solution la plus coûteuse en termes de licence, mais elle se démarque clairement par un **impact business élevé** (8–10) et une **réduction importante du risque cyber** (10), grâce à ses fonctionnalités de sécurité avancées.
- HAProxy Enterprise** apparaît comme une option intermédiaire : les coûts sont mieux équilibrés entre licence, technique et humain, mais son **impact**

business reste plus limité.

- **BunkerWeb**, en revanche, bénéficie d'un **coût de licence faible (open source)**, mais les **efforts humains et techniques nécessaires** sont plus importants, avec au final un **impact business assez réduit**.

Après avoir pris en compte l'ensemble des critères et résultats de l'étude, le choix principal proposé à l'équipe a été **Imperva WAF**. Un premier atelier a même été organisé avec le service concerné pour approfondir l'évaluation. **HAProxy** reste également une option étudiée. Pour le moment, le projet est **mis en pause**, en attendant la validation du budget et les retours des deux services impliqués.

B. Étude comparative des solutions de partage sécurisé de secrets

1. Contexte et objectif

L'objectif de cette étude est de sélectionner une solution permettant le **partage temporaire et sécurisé de secrets** (mots de passe, fichiers sensibles) dans un environnement interne, tout en respectant les exigences suivantes :

- Confidentialité et chiffrement
- Intégration dans l'infrastructure : Docker, Kubernetes, CI/CD.
- Expiration et suppression automatique des secrets
- Supervision et observabilité (logs et monitoring)

Deux solutions ont été comparées et testées :

- **Cryptgeon [6]**
- **Yopass [7]**

Une troisième solution (Onetimesecret) a été analysée mais n'a pas été retenue pour les tests approfondis(Non souveraine).

2. Méthodologie de l'étude

L'étude a suivi une approche structurée :

- **Définition des critères de sélection**, couvrant les aspects financiers, juridiques, fonctionnels, techniques, opérationnels et de durabilité.

- **Réalisation d'un PoC** pour chaque solution afin de tester le déploiement, les fonctionnalités et l'intégration interne.
- **Analyse documentaire et technique** : GitHub, documentation officielle, guides Docker/Kubernetes, forums, rapports existants.
- **Production de livrables** : documentation, tableau comparatif, rapports techniques, et présentation synthétique pour l'équipe.
- **Présentation et discussion** lors d'une réunion d'équipe pour valider les choix.

3. Synthèse comparative

- **Tableau comparatif des solutions – résumé exécutif**

Ce tableau présente un résumé exécutif des principales solutions de partage sécurisé de secrets étudiées, en mettant en avant les critères essentiels retenus pour l'évaluation.

| Catégorie | Critères | Yopass | Cryptgeon |
|-----------------------|---|------------------------------|---------------|
| Financier & Juridique | Open Source, conformité RGPD/UE | Oui | Oui |
| Fonctionnel | Partage temporaire et sécurisé | Oui (CLI pour CI/CD) | Oui |
| Technique | Chiffrement côté client, Docker/K8s | Oui | Oui |
| Opérationnel | Déploiement on-premise, documentation, mises à jour | Oui | Oui |
| Sécurité | Chiffrement, expiration automatique, monitoring | Oui (Prometheus/OpenMetrics) | Oui (basique) |
| Durabilité | Popularité / communauté | Forte | Moyenne |

Tableau 05: Tableau comparatif des solutions de partages de secrets

Une fois les livrables finalisés, j'ai présenté les résultats à l'équipe via un PowerPoint explicatif. Le manager a validé la solution Yopass.

Motifs du choix : intégration CLI facilitant l'utilisation dans les pipelines CI/CD, monitoring natif via Prometheus/OpenMetrics, déploiement flexible avec Docker/Kubernetes, et communauté active.

À l'heure actuelle, lors de la rédaction de ce rapport, je procède à son implémentation au niveau de l'environnement de recette afin de réaliser les premiers tests.

C. Apports

Pour l'entreprise:

- **Renforcement de la sécurité et de la conformité :** sélection d'Imperva WAF pour la protection des applications web et de Yopass pour le partage sécurisé de secrets.
- **Base solide pour le déploiement :** réalisation d'un PoC et production de livrables documentés, facilitant l'intégration et l'adoption progressive des solutions dans les environnements de recette et de production.

Pour moi:

- Réalisation de PoC, déploiement, tests de solutions et analyse comparative de solutions WAF et de partage sécurisé de secrets,
- Production de livrables structurés (Excel, PowerPoint) et présentation des résultats à l'équipe,
- Découverte du rôle et des responsabilités d'un architecte sécurité cloud, notamment dans l'évaluation et la sélection de solutions adaptées aux besoins d'une organisation.

4. Apports de l'expérience en entreprise

Au terme de cette alternance, l'ensemble des missions réalisées m'a permis de développer des compétences techniques, méthodologiques et relationnelles.

Sur le plan technique, j'ai consolidé mes connaissances en sécurité applicative, en explorant différentes approches de tests et en expérimentant leur intégration dans des environnements cloud et des pipelines CI/CD complexes. Ces expériences m'ont également sensibilisé aux enjeux liés à la sécurité de la chaîne d'approvisionnement logicielle, à la conformité réglementaire (ISO 27001, SecNumCloud, HDS), ainsi qu'à la sélection et à l'évaluation de solutions de sécurité dans un contexte opérationnel.

Au-delà de l'aspect technique, cette alternance m'a offert un cadre d'apprentissage important sur le plan méthodologique et relationnel. J'ai appris à présenter mes travaux devant une équipe complète et à défendre mes choix techniques avec rigueur. Ces situations ont renforcé mes compétences relationnelles : communication, pédagogie, esprit d'équipe, et confiance en moi pour intervenir lors d'analyses en direct.

J'ai également pris pleinement conscience de l'importance de la documentation professionnelle pour assurer la compréhension, la traçabilité et la pérennité des solutions mises en place.

Enfin, cette expérience m'a permis de me familiariser avec le rôle d'architecte sécurité cloud, en comprenant ses responsabilités : anticiper les menaces, proposer des solutions adaptées, et accompagner leur adoption au sein de l'organisation. Elle m'a donné confiance dans ma capacité à analyser, synthétiser et présenter des solutions complexes, tout en tenant compte de leurs impacts financiers et opérationnels.

5. Conclusion

Au terme de cette alternance, je peux affirmer que mes objectifs initiaux ont été atteints en grande partie. J'ai renforcé mes compétences techniques en sécurité cloud et applicative, si je dispose désormais de bases solides, je suis conscient qu'il reste un travail d'approfondissement à mener pour exceller pleinement dans ce domaine. Cette perspective constitue pour moi une motivation supplémentaire à poursuivre mes efforts et à continuer de progresser.

Cette expérience m'a apporté bien plus qu'une mise en pratique des connaissances acquises au cours de ma formation. Elle m'a permis d'élargir mon champ de vision en explorant des domaines de la sécurité peu abordés dans le cadre académique notamment la sécurité des environnements cloud et l'approche DevSecOps. J'ai pu mobiliser mes connaissances théoriques pour comprendre et mettre en œuvre des solutions concrètes. J'ai également appris à travailler en équipe, à communiquer mes idées et à m'adapter à un environnement professionnel réel.

En particulier, mon immersion dans des projets orientés DevSecOps et cloud m'a permis d'acquérir une compréhension complète des infrastructures modernes, de leur conception jusqu'à leur sécurisation. Cette base me sera précieuse pour atteindre mon objectif professionnel : intégrer une Red Team spécialisée dans ces technologies. Je suis convaincu que pour identifier et exploiter efficacement les failles de sécurité, il est indispensable de maîtriser le fonctionnement interne des environnements que l'on évalue. Mon ambition est donc de capitaliser sur cette expérience pour évoluer vers un rôle d'expert en sécurité offensive, tout en continuant à me former et à perfectionner mes compétences.

Comme le disait *Albert Einstein* : « *L'expérience, c'est le nom que chacun donne à ses erreurs.* » Cette alternance m'a offert l'opportunité d'apprendre, d'expérimenter et de progresser dans un environnement concret, en me préparant aux défis futurs de ma carrière.

Bibliographie

- **Normes et standards**

[1] ISO/IEC 27001 – Information Security Management Systems (ISMS) – Requirements.

[2] ANSSI – SecNumCloud : Référentiel de sécurité pour les services cloud certifiés.

[3] HDS – Hébergement de Données de Santé : normes et recommandations officielles.

- **Outils et solutions**

[4] OWASP ZAP. Official Documentation. <https://www.zaproxy.org/>

[5] Imperva WAF. Documentation produit. <https://www.imperva.com/>

[6] Cryptgeon – Documentation officielle, <https://cryptgeon.com>

[7] Yopass. Documentation officielle et guides d'utilisation. <https://yopass.se/>.

[8] Docker. Guides officiels de sécurisation des conteneurs et images. <https://www.docker.com/>

[9] Trivy. Scanner de vulnérabilités pour conteneurs. <https://aquasec.com/trivy>

[10] Cosign. Documentation sur la signature et vérification d'artefacts. <https://docs.sigstore.dev/cosign>

[11] MinIO. Documentation officielle sur le stockage d'objets compatibles S3. <https://min.io/>

[12] Harbor. Documentation officielle sur le registre de conteneurs sécurisé. <https://goharbor.io/>

- **Référentiels et tutoriels**

GitHub repositories officiels pour OWASP ZAP, Trivy, Cosign, MinIO, yopass, cryptgeon et Harbor consultés pour PoC et tests pratiques.