Rapport de projet

De Martino Giada Flora, Dorian Tetu

Table of Contents

Introduction

Beep est une application de messagerie sociale permettant aux utilisateurs de **communiquer, créer des communautés et interagir en temps réel**. Elle propose une expérience complète mêlant messagerie instantanée, appels audio/vidéo, partage de fichiers et gestion collaborative de serveurs (espaces de discussion).

Son architecture repose sur une infrastructure distribuée intégrant un backend robuste, une interface utilisateur réactive, ainsi qu'un service de messagerie électronique basé sur le protocole SMTP.

Ce rapport détaille les résultats de ces analyses, les risques identifiés, ainsi que les actions correctives recommandées pour renforcer la sécurité globale du projet.

Diagramme d'architecture

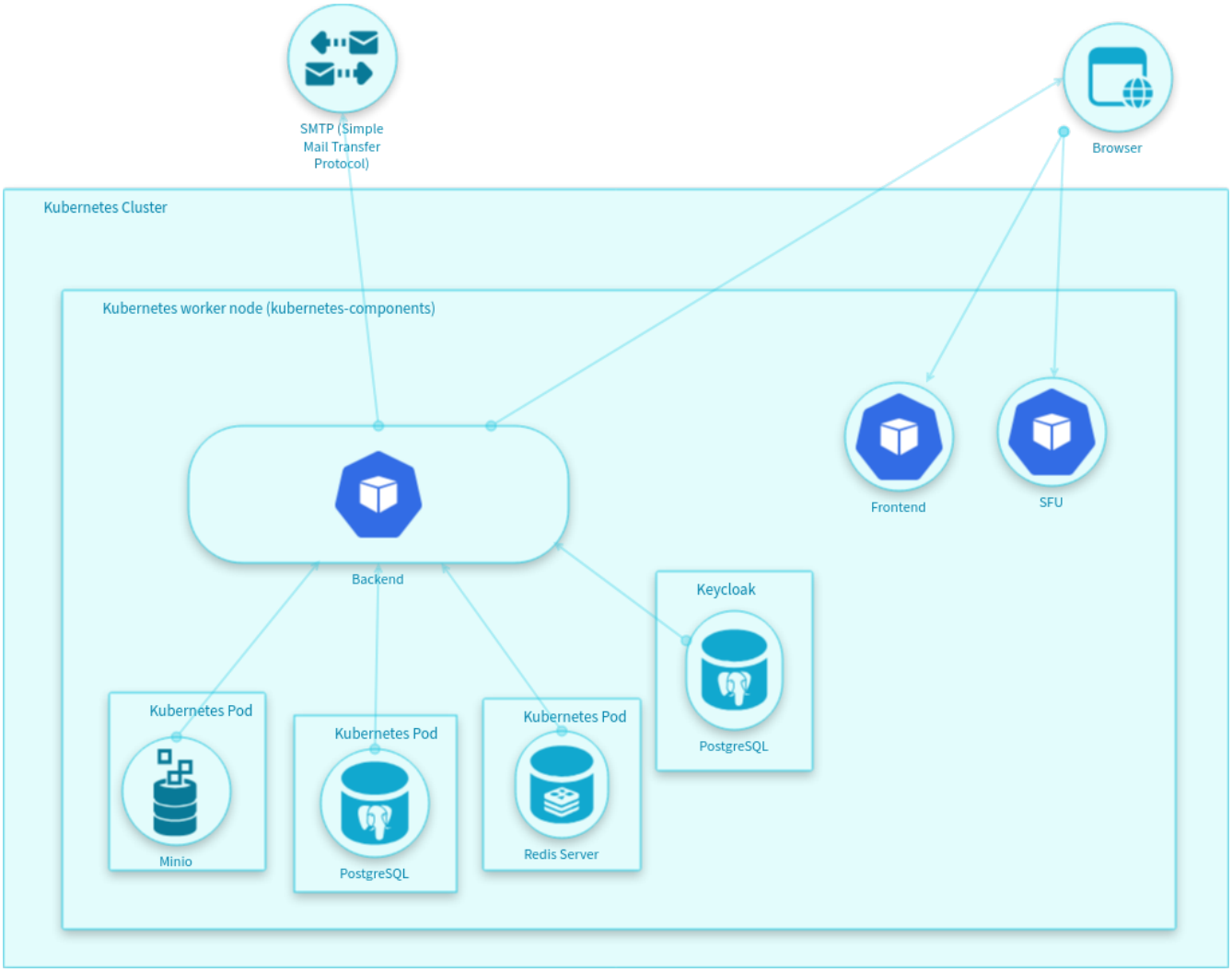

Figure 1. Diagramme d'architecture du projet Beep

Pourquoi effectuer des scans de sécurité

L'objectif des scans de sécurité est de détecter les vulnérabilités potentielles dans l'application afin

de renforcer sa résilience face aux attaques. Deux types de scans ont été réalisés :

- **SAST (Static Application Security Testing)** : Analyse du code source pour identifier les failles de sécurité avant l'exécution. Cette méthode permet de cibler les failles dès les premières phases du développement.
- **Scan Trivy** : Analyse ciblée des configurations de l'infrastructure, notamment du service SMTP, qui est un vecteur critique dans le contexte d'une application de messagerie.

Ces analyses permettent de prendre des mesures proactives avant la mise en production, de se conformer aux bonnes pratiques de sécurité logicielle, et de limiter les risques liés à l'exploitation de failles connues.

Résultats des scans

SAST IriusRisk

Voici les cinq **threats** les plus critiques identifiés dans l'application Beep, principalement liés à la gestion du protocole SMTP :

- **Threat 1** : `SMTP - STARTTLS non activé` → Le chiffrement des communications SMTP via STARTTLS est essentiel pour éviter l'interception des messages par des tiers. L'absence de cette couche rend les communications vulnérables à l'écoute passive (attaque de type **man-in-the-middle**).
- **Threat 2** : `SMTP - Absence de rate limiting` → Sans limitation du nombre de messages envoyés par IP ou utilisateur, l'infrastructure est exposée à des attaques par saturation (flood), pouvant impacter la disponibilité du service.
- **Threat 3** : `SMTP - Configuration non auditée régulièrement` → Des erreurs ou oublis dans la configuration peuvent introduire des vulnérabilités. Des audits réguliers automatisés permettraient de détecter rapidement tout changement ou mauvaise configuration.
- **Threat 4** : `SMTP - Absence de contrôle d□authenticité des e-mails` → Le manque d'authentification des e-mails via SPF, DKIM et DMARC expose les utilisateurs à des tentatives de phishing et à l'usurpation d'identité.
- **Threat 5** : `SMTP - Relais ouvert` → Laisser le serveur SMTP agir comme relais non authentifié permet à des tiers d'envoyer des courriers frauduleux, ce qui peut faire inscrire le domaine sur des listes noires (blacklists).

SAST Snyk

Le scan Snyk s'est concentré sur les dépendances et la sécurité du code. Les principales vulnérabilités sont les suivantes :

- **Vulnérabilité 1** : `Dockerfile - Version de Node obsolète` → La version utilisée présentait des failles critiques. Il est recommandé d'utiliser au minimum la version `22.16.0` de Node.js.
- **Vulnérabilité 2** : `Hardcoded Secret` → Un secret de chiffrement était codé en dur dans le code source (`jsonwebtoken.default.sign`). Ce type de pratique est à proscrire. Il faut utiliser une

variable d'environnement (`APP_KEY`) définie dans un fichier `.env` sécurisé.

- D'autres vulnérabilités incluent l'utilisation peu sécurisée de certaines fonctions comme `send()`, qui, si mal paramétrée, peut divulguer des données sensibles, notamment en cas de mauvaise gestion des erreurs.

GitHub Action CI

L'intégration continue (CI) via GitHub Actions permet d'automatiser les tests et les analyses de sécurité. Les résultats des scans exécutés dans ce cadre indiquent :

- **Vulnérabilité 1** : `Cross-spawn - Version obsolète` → La version utilisée doit être mise à jour vers `7.0.5` pour corriger des failles connues et améliorer la stabilité générale des workflows CI.

Exploitation des résultats et plan d'action

Les résultats obtenus orientent la sécurisation du serveur SMTP, qui constitue un point d'entrée particulièrement sensible dans l'architecture de Beep. Un plan d'action structuré, par priorité, est indispensable pour assurer la sécurité globale du système.

Tâches à prioriser immédiatement (urgence haute)

1. **Activer STARTTLS** : Implémenter et forcer STARTTLS sur le serveur SMTP pour assurer le chiffrement des e-mails.
2. **Mettre à jour les dépendances critiques** :
   - Mettre à jour Node.js (`22.16.0`).
   - Mettre à jour `Cross-spawn`.
   - Corriger toutes les dépendances listées par Snyk et Trivy.
3. **Remplacer les secrets codés en dur** : Référencer les clés dans un fichier `.env`, avec gestion sécurisée via un coffre de secrets (Vault ou équivalent).

Tâches à mener à court terme

1. **Implémenter une politique de rate limiting** : Protéger le service SMTP contre les abus en définissant des quotas par utilisateur/IP.
2. **Configurer l'authentification des e-mails** : Ajouter les enregistrements DNS nécessaires pour activer SPF, DKIM et DMARC. Cela contribuera aussi à la délivrabilité des messages.
3. **Désactiver tout relais SMTP non autorisé** : Vérifier que seul le trafic authentifié est autorisé à transiter par le serveur.

Tâches à intégrer à moyen terme

1. **Mettre en place des audits réguliers de configuration** : Automatiser les contrôles de configuration SMTP (via Ansible, OpenSCAP ou des scripts internes).

2. **Renforcer la CI/CD avec des règles de sécurité strictes** :
    - Ajouter des vérifications dans la pipeline CI pour détecter les secrets accidentellement poussés.
    - Intégrer des scans réguliers de vulnérabilités avant chaque déploiement.
3. **Documenter** : Rédiger des bonnes pratiques de sécurité sur la gestion des secrets et la configuration sécurisée des services.

Conclusion

La sécurisation de l'application Beep passe par une vigilance constante, dès les phases de développement jusqu'à la mise en production. Les résultats des analyses montrent une exposition critique du serveur SMTP, mais également des problèmes dans les dépendances logicielles et la gestion des secrets.

Grâce aux actions proposées, il est possible de réduire considérablement la surface d'attaque en quelques étapes ciblées. À terme, nous visons non seulement la correction des vulnérabilités actuelles, mais également l'instauration d'une culture de sécurité continue, intégrée à nos processus DevOps. La priorisation des tâches en trois phases permet une montée en sécurité progressive, tout en assurant la stabilité du projet.

Un suivi régulier des actions, ainsi que la mise en place de métriques de sécurité, permettront de s'assurer de l'efficacité des mesures appliquées et d'anticiper les menaces futures.