CONFIDENTIEL

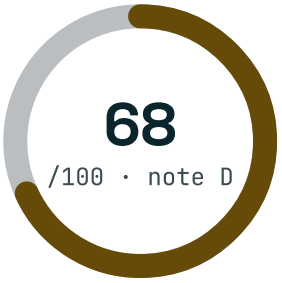
CyberSure CYBERSECURITY

Cybersécurité applicative · Belgique

RAPPORT D'AUDIT DE SÉCURITÉ

Portail Client & Core API v2.4

Acme Corp Europe (Spécimen anonymisé)



Score : **68/100** · note **D** vérifié par analyse

Constats : 9 · 2 critiques · 3 élevés

Outils : Gitleaks, Semgrep, Trivy, ShellCheck

DATE	AUDITEUR	ENTITÉ	CONTACT
2026-11-18	Nicola Alfè	BCE BE 1005.721.833	info@cybersure.be

Audit réalisé en local dans un environnement cloisonné · aucun secret conservé ni partagé

Comprendre les résultats et la direction à prendre

L'examen de l'application a fait ressortir 7 sujet(s) distinct(s) à examiner ou à améliorer, répartis sur 7 signalement(s). Plusieurs signalements peuvent avoir une même cause et être corrigés ensemble. 5 sujet(s) sont classés prioritaires : ils doivent être vérifiés en premier afin de concentrer les efforts sur les risques les plus importants.

Des clés ou mots de passe pourraient donner accès à des services s'ils sont réels et encore utilisables. Certaines informations saisies ou importées pourraient être interprétées comme des instructions plutôt que comme de simples données. Des composants utilisés par l'application sont associés à des problèmes de sécurité connus ; leur impact dépend de la version et de l'usage réel.

L'application étant accessible sur Internet, les points pouvant être atteints depuis l'extérieur doivent être examinés en premier. La présence de plusieurs organisations ou de données sensibles renforce l'importance de vérifier les accès.

Améliorations à engager

1. D'abord — confier les sujets prioritaires à un responsable technique : vérifier les fichiers cités, distinguer un problème réel d'un exemple ou d'un test et noter la décision. Si un accès privé réellement exposé est confirmé, le remplacer sans attendre ; limiter temporairement les fonctions exposées uniquement si le risque le justifie.
2. Accès et clés — demander au responsable de vérifier chaque valeur en privé, de révoquer les accès réels exposés et de retirer les valeurs des fichiers distribués. Attendre une preuve de remplacement et de révocation, sans recopier les clés.
3. Saisies et affichage — faire corriger les traitements signalés pour que les informations reçues restent des données, et non des instructions. Demander un test de saisie inhabituelle et un test de fonctionnement normal pour chaque correction.
4. Composants — faire confirmer les versions concernées et les correctifs dans les avis officiels, mettre à jour puis tester les fonctions utilisées. Sans correctif disponible, convenir d'une protection temporaire et d'une date de réexamen.
5. Enfin — attribuer une échéance et un responsable à chaque correction, conserver les résultats des tests, puis refaire les contrôles concernés. Clore un sujet seulement après vérification ; pour un risque non corrigé, documenter la décision et sa protection temporaire.
6. Compléter les contrôles qui n'ont pas pu aboutir avant de considérer le périmètre comme suffisamment examiné.

Ce que ce rapport permet de conclure : Les résultats tiennent compte de l'analyse enregistrée et des arbitrages éventuels. 2 remarque(s) signalent des contrôles incomplets ou des vérifications encore nécessaires ; ce ne sont pas autant de failles confirmées. Les conséquences décrites sont possibles, pas la preuve d'une attaque ou d'une fuite déjà survenue.

CONTEXTE MÉTIER

Exposition : Internet public · Données : Financières · Authentification : Multi-tenant

Application SaaS B2B exposée sur Internet, gérant des transactions bancaires et des données de facturation multi-organisations.

SOMMAIRE

1. Synthèse et indicateurs
2. Analyse approfondie et angles morts
3. Plan d'action priorisé
4. Constats détaillés et remédiations
5. Méthodologie et confidentialité
6. Vérifications manuelles ciblées

SYNTHÈSE POUR LA DIRECTION

Le niveau de risque global de l'application est critique. 7 problème(s) distinct(s) ont été relevés, dont 5 à traiter en priorité. Une correction rapide est recommandée avant toute mise en production ou nouvelle diffusion.

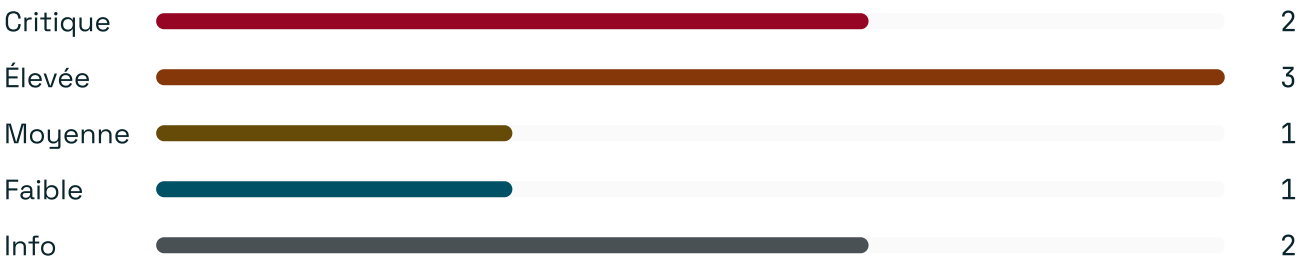
- Risque de fuite d'accès (clés, mots de passe) permettant une prise de contrôle de services.
- Risque de vol ou de modification de la base de données clients.
- Risque lié à des composants tiers obsolètes avec failles publiques connues.

Effort de correction estimé : 3 à 4.5 jour(s) de développement.

COMPARAISON AVEC UN AUDIT PRÉCÉDENT

01 Synthèse et indicateurs

RÉPARTITION PAR SÉVÉRITÉ



FICHIERS ANALYSÉS

214

TOTAL CONSTATS

9

OUTILS EXÉCUTÉS

Gitleaks · Semgrep · Trivy · ShellCheck

RÉPARTITION OWASP TOP 10

A03:2021 - Injection	3 · 43 %
A07:2021 - Identification and Authentication Failures	1 · 14 %
A05:2021 - Security Misconfiguration	1 · 14 %
A06:2021 - Vulnerable and Outdated Components	1 · 14 %
A02:2021 - Cryptographic Failures	1 · 14 %

FICHIERS LES PLUS EXPOSÉS (HOTSPOTS)

71 % des constats réels se concentrent dans ces 5 fichier(s), sur 7 fichier(s) concerné(s).

1.	src/config.ts	1
2.	api/users.php	1
3.	src/components/Comment.tsx	1
4.	server/index.ts	1
5.	package-lock.json	1

02 Analyse approfondie et angles morts

Synthèse de l'analyse approfondie

L'analyse approfondie confirme 3 vulnérabilités majeures nécessitant une correction immédiate ou à court terme : une clé secrète de production Stripe exposée en clair (P0), une injection SQL exploitable sur l'export des utilisateurs (P0), et un risque de XSS stockée dans le rendu des commentaires (P1).

Les alertes détectées dans les dossiers de test (notamment les clés de simulation) ont été requalifiées en faux positifs. L'application présente par ailleurs une bonne isolation globale entre les organisations, mais la révocation effective des sessions après déconnexion doit être renforcée.

`confirmés : 4` `probables : 0` `faux positifs : 0`

CONTRÔLES MANQUANTS / ANGLES MORTS

- Configurations Cloud de production : seuls les fichiers de code et le Dockerfile ont été audités ; les règles de pare-feu applicatif (WAF) et les en-têtes réseau réels doivent être vérifiés sur l'infrastructure déployée.
- Traçabilité dynamique des flux inter-services : l'analyse statique locale ne remplace pas une vérification continue des communications API externes.

03 Plan d’action priorisé

P0 1. Révocation et rotation immédiate de la clé Stripe live

effort : 1 heure

Révoquer la clé dans le tableau de bord Stripe. Configurer la variable d’environnement STRIPE_SECRET_KEY sur le serveur de production.

constats : #1

P0 2. Sécurisation de la requête SQL d’utilisateurs (PDO préparé)

effort : 2 heures

Modifier api/users.php pour utiliser des requêtes préparées avec typage strict de l’identifiant.

constats : #2

P1 3. Assainissement du rendu HTML des commentaires (DOMPurify)

effort : 3 heures

Ajouter la bibliothèque DOMPurify et assainir systématiquement le contenu HTML avant affichage dans Comment.tsx.

constats : #3

P1 4. Restriction de la politique CORS sur l’API Express

effort : 1 heure

Remplacer le joker ’*’ par les noms de domaine précis du portail client dans server/index.ts.

constats : #4

Matrice de criticité qualitative

Appréciation de l’auditeur : exploitation × impact métier. Ni score CVSS ni calcul OWASP Risk Rating normalisé.

Exploitation / Impact	Faible	Modéré	Critique
Facile	Moyenne	Élevée	Critique
Moyenne	Faible	Moyenne	Élevée
Difficile	Faible	Faible	Moyenne

04 Constats détaillés et remédiations — 9 problème(s) distinct(s), 9 occurrence(s)

#1 Secret Exposé : Stripe Access Token

P0 · confirmé CRITIQUE

outil: Gitleaks règle: stripe-access-token

CWE-798: Use of Hard-coded Credentials

A07:2021 - Identification and Authentication Failures

Un secret de type « stripe-access-token » a été détecté en clair dans le code source.

EMPLACEMENTS CONCERNÉS · 1 OCCURRENCE(S), 1 FICHIER(S)

src/config.ts — ligne(s) 4 · 1 occurrence(s)

```
const STRIPE_KEY = "sk_live_51MzREDACTED_8f3d928374";
```

VÉRIFICATION DE L'ANALYSE

Verdict : Clé secrète de production Stripe active présente en clair dans src/config.ts. Exploitation immédiate possible par quiconque accède au code.

Impact réel : Exécution d'opérations de débit, de remboursement et de virement non autorisées via l'API Stripe.

Correction : Révoquer la clé dans le tableau de bord Stripe, générer un nouveau secret et le charger via process.env.STRIPE_SECRET_KEY.

```
export const STRIPE_KEY = process.env.STRIPE_SECRET_KEY;
```

ARBITRAGE DE L'AUDITEUR

Auditeur : Confirmé · Critique

Clé secrète de production active Stripe en clair dans la configuration. Risque d'opérations bancaires directes non autorisées.

Matrice qualitative : **Critique** · Facile / Critique

2026-11-18 10:30

REMÉDIATION RECOMMANDÉE

Impact possible : Une valeur privée encore active et accessible à un tiers pourrait lui permettre d'utiliser le service avec les droits de son propriétaire.

ACTIONS À EFFECTUER

1. Au fichier et à la ligne indiqués, identifier confidentiellement le fournisseur et le type de valeur : exemple, clé publique, clé de test ou secret privé. Ne pas recopier la valeur dans un ticket.
2. Si le secret privé est réel et exposé, le révoquer dans la console du fournisseur, créer un remplacement avec les seuls droits nécessaires et le charger depuis un coffre ou une variable d'environnement côté serveur.
3. Retirer la valeur du code, des fichiers distribués et des journaux ; si elle est dans Git, coordonner le nettoyage de l'historique avec l'équipe après révocation. Ajouter Gitleaks à la vérification des prochains changements.

Conditions à vérifier : Une alerte seule ne prouve ni que la clé est privée, ni qu'elle est active. Confirmer son propriétaire, ses permissions et son exposition sans l'envoyer à un service tiers.

Validation après correction : Contrôler dans la console du fournisseur que l'ancienne valeur est révoquée ; si un test d'accès est autorisé, utiliser une opération de lecture non destructive et vérifier son refus (code selon le fournisseur, pas nécessairement 401). Vérifier que le remplacement fonctionne puis relancer Gitleaks.

#2 Entrée Utilisateur Concaténée Dans Une Requête SQL

P0 · confirmé CRITIQUE

outil: Semgrep règle: php.lang.security.injection.tainted-sql-string

CWE-89: SQL Injection A03:2021 - Injection

Entrée utilisateur issue de $_GET['id'] concaténée directement dans une requête SQL sans paramétrage : risque critique d'injection SQL.

EMPLACEMENTS CONCERNÉS · 1 OCCURRENCE(S), 1 FICHIER(S)

api/users.php — ligne(s) 17 · 1 occurrence(s)

```
$db->query("SELECT * FROM users WHERE id=" . $_GET['id']);
```

VÉRIFICATION DE L'ANALYSE

Verdict : Concaténation directe de $_GET['id'] dans la chaîne de requête SQL. Aucune validation ou assainissement en amont.

Impact réel : Exfiltration complète de la base de données utilisateurs (emails, rôles, empreintes de mot de passe).

Correction : Remplacer par une requête préparée PDO avec paramètre lié typé (int).

```
$stmt = $db->prepare('SELECT id, email, role FROM users WHERE id = :id');
$stmt->execute(['id' => (int)$_GET['id']]);
$user = $stmt->fetch(PDO::FETCH_ASSOC);
```

ARBITRAGE DE L'AUDITEUR

Auditeur : Confirmé · Critique

Injection SQL directe via paramètre GET sur l'export utilisateurs. Extraction intégrale de la base de données possible.

Matrice qualitative : **Critique** · Facile / Critique

2026-11-18 10:35

REMÉDIATION RECOMMANDÉE

Impact possible : Si une entrée externe atteint une requête SQL concaténée, elle pourrait modifier la requête et lire ou altérer des données hors des droits prévus.

ACTIONS À EFFECTUER

1. À l'emplacement indiqué, séparer le texte SQL des valeurs : utiliser les paramètres du pilote ou l'ORM, jamais une concaténation ou une interpolation d'entrée.
2. Pour les noms de colonnes et les tris (non paramétrables comme des valeurs), mapper une liste fermée de choix vers des identifiants SQL fixes.
3. Valider les types et limiter les droits du compte SQL ; la validation d'entrée ne remplace pas la requête paramétrée.

Avant — exemple illustratif, pas le code extrait du projet

```
$pdo->query("SELECT * FROM users WHERE name = '" . $name . "'");
```

Après — exemple à adapter et tester dans le projet

```
$stmt = $pdo->prepare('SELECT * FROM users WHERE name = :name');
$stmt->execute(['name' => $name]);
```

Conditions à vérifier : Suivre la valeur de la requête ou de l'import jusqu'à execute/query. Vérifier les paramètres déjà utilisés avant de qualifier B608 ou une règle SQLi.

Validation après correction : En base de test, rechercher une valeur contenant une apostrophe, par exemple O'Connor ou O'Connor : elle doit rester une donnée, sans erreur de syntaxe ni retour de lignes supplémentaires. Tester un tri non autorisé : il doit être refusé. Vérifier aussi une recherche normale.

#3 Détection De DangerouslySetInnerHTML Avec Donnée Non Assainie

P1 · confirmé ÉLEVÉE

outil: Semgrep règle: typescript.react.security.audit.react-dangerouslysetinnerhtml

CWE-79: Improper Neutralization of Input During Web Page Generation

A03:2021 - Injection

Attribut dangerouslySetInnerHTML alimenté par une propriété sans assainissement préalable : risque de Cross-Site Scripting (XSS).

EMPLACEMENTS CONCERNÉS · 1 OCCURRENCE(S), 1 FICHIER(S)

src/components/Comment.tsx — ligne(s) 42 · 1 occurrence(s)

```
<div dangerouslySetInnerHTML={{ __html: comment.body }} />
```

VÉRIFICATION DE L'ANALYSE

Verdict : dangerouslySetInnerHTML injecte le corps du commentaire sans filtrage HTML. Permet l'exécution de code JavaScript arbitraire dans le navigateur des visiteurs.

Impact réel : Vol de jetons de session et usurpation de compte d'utilisateurs consultant la section des commentaires.

Correction : Assainir la chaîne avec DOMPurify ou utiliser un composant de rendu Markdown sécurisé sans HTML brut.

```
import DOMPurify from 'dompurify';
<div dangerouslySetInnerHTML={{ __html: DOMPurify.sanitize(comment.body) }} />
```

ARBITRAGE DE L'AUDITEUR

Auditeur : Confirmé · Élevée

Rendu HTML brut dans les commentaires sans assainissement. Risque de vol de cookies de session.

Matrice qualitative : **Moyenne** · Moyenne / Modéré

2026-11-18 10:38

REMÉDIATION RECOMMANDÉE

Impact possible : Si du HTML non fiable est rendu sans protection adaptée, il pourrait exécuter des actions dans le navigateur d'un lecteur avec les droits de sa session.

ACTIONS À EFFECTUER

1. Retrouver la source du texte et le contexte de rendu (texte HTML, attribut, URL, courriel). Pour du texte simple, utiliser le rendu échappé du framework, pas innerHTML ni un marqueur safe.
2. Si le HTML riche est indispensable, appliquer un assainisseur maintenu avec une liste fermée de tags/attributs autorisés juste avant le rendu. Ne pas neutraliser ensuite cette protection par une concaténation.
3. Vérifier les rendus d'exports et de courriels autant que les pages ; une politique CSP est une défense supplémentaire, pas un remplacement de l'encodage.

Avant — exemple illustratif, pas le code extrait du projet

```
<div dangerouslySetInnerHTML={{ __html: value }} />
```

Après — exemple à adapter et tester dans le projet

```
// Texte simple, échappé par React
<div>{value}</div>
```

Conditions à vérifier : Confirmer qu'un utilisateur ou un fichier importé contrôle la valeur et qu'aucun assainissement adapté n'est effectué sur le chemin réellement exécuté.

Validation après correction : Dans un environnement autorisé, enregistrer test et des guillemets : le champ texte doit les afficher littéralement. Pour du HTML riche, vérifier la suppression des attributs on* et des URL javascript: sans les exécuter, puis vérifier que les tags permis restent disponibles.

#4 CORS Autorise Toutes Les Origines Avec Credentials

P1 · confirmé ÉLEVÉE

outil: Semgrep règle: javascript.express.security.cors-misconfiguration

CWE-942: Permissive Cross-domain Policy with Untrusted Domains

A05:2021 - Security Misconfiguration

Configuration CORS autorisant origin: '*' simultanément avec credentials: true. Risque d'interception de requêtes de session par des sites tiers.

EMPLACEMENTS CONCERNÉS · 1 OCCURRENCE(S), 1 FICHIER(S)

server/index.ts — ligne(s) 23 · 1 occurrence(s)

```
app.use(cors({ origin: '*', credentials: true }))
```

VÉRIFICATION DE L'ANALYSE

Verdict : CORS configuré avec origin: '*' et credentials: true. Incohérence de sécurité qui permet à des domaines tiers d'émettre des requêtes authentifiées.

Impact réel : Attaques par Cross-Origin Resource Sharing permettant le détournement de requêtes utilisateur.

Correction : Définir une liste blanche stricte des origines frontend autorisées.

```
app.use(cors({ origin: ['https://app.acme.com'], credentials: true }));
```

ARBITRAGE DE L'AUDITEUR

REMÉDIATION RECOMMANDÉE

Impact possible : Une valeur privée encore active et accessible à un tiers pourrait lui permettre d'utiliser le service avec les droits de son propriétaire.

ACTIONS À EFFECTUER

1. Au fichier et à la ligne indiqués, identifier confidentiellement le fournisseur et le type de valeur : exemple, clé publique, clé de test ou secret privé. Ne pas recopier la valeur dans un ticket.
2. Si le secret privé est réel et exposé, le révoquer dans la console du fournisseur, créer un remplacement avec les seuls droits nécessaires et le charger depuis un coffre ou une variable d'environnement côté serveur.
3. Retirer la valeur du code, des fichiers distribués et des journaux ; si elle est dans Git, coordonner le nettoyage de l'historique avec l'équipe après révocation. Ajouter Gitleaks à la vérification des prochains changements.

Conditions à vérifier : Une alerte seule ne prouve ni que la clé est privée, ni qu'elle est active. Confirmer son propriétaire, ses permissions et son exposition sans l'envoyer à un service tiers.

Validation après correction : Contrôler dans la console du fournisseur que l'ancienne valeur est révoquée ; si un test d'accès est autorisé, utiliser une opération de lecture non destructive et vérifier son refus (code selon le fournisseur, pas nécessairement 401). Vérifier que le remplacement fonctionne puis relancer Gitleaks.

#5 CVE-2024-38096 — Axios

ÉLEVÉE

outil: Trivy règle: trivy.CVE-2024-38096

`CWE-1333: Inefficient Regular Expression Complexity`

`A06:2021 - Vulnerable and Outdated Components`

Déni de service par expression régulière lors du traitement de requêtes avec en-têtes volumineux.

EMPLACEMENTS CONCERNÉS · 1 OCCURRENCE(S), 1 FICHIER(S)

`package-lock.json` — ligne non fournie · 1 occurrence(s)

ARBITRAGE DE L'AUDITEUR

REMÉDIATION RECOMMANDÉE

Impact possible : Le composant est associé à un avis de sécurité ; l'impact réel dépend de la version installée et de l'utilisation de la fonction affectée.

ACTIONS À EFFECTUER

1. Paquet signalé : axios ; version installée rapportée : 0.21.1 ; correctif(s) rapporté(s) : 0.21.2. Vérifier ces valeurs dans l'avis officiel avant mise à jour.
2. Comparer le paquet, sa version installée et la plage affectée à l'avis officiel. En cas de branches correctives contradictoires, ne pas choisir arbitrairement une version.
3. Mettre à jour le manifeste via le gestionnaire de paquets et régénérer le verrouillage. Pour un paquet indirect, identifier le parent (npm explain, pip show ou composer why) et mettre à jour ce parent.
4. Sans correctif confirmé, désactiver la fonction affectée ou limiter son exposition selon l'avis ; documenter responsable, mesure et échéance.

Conditions à vérifier : Une plage corrective n'est pas une version installée. Vérifier le verrouillage et l'environnement déployé, pas seulement le manifeste.

Validation après correction : Exécuter les tests applicatifs, vérifier la version réellement installée puis relancer le scanner concerné (npm audit, composer audit, pip-audit ou Trivy). Le même avis doit disparaître ou être explicitement traité par une mesure compensatoire, sans régression fonctionnelle.

#6 Math.Random() N'est Pas Cryptographiquement Sûr

MOYENNE

outil: Semgrep règle: javascript.lang.security.audit.insecure-random

CWE-338: Use of Cryptographically Weak PRNG A02:2021 - Cryptographic Failures

Math.random() est un générateur pseudo-aléatoire prédictible. Ne pas utiliser pour la génération de jetons d'authentification ou de réinitialisation.

EMPLACEMENTS CONCERNÉS · 1 OCCURRENCE(S), 1 FICHIER(S)

src/lib/token.ts — ligne(s) 8 · 1 occurrence(s)

```
const token = Math.random().toString(36).slice(2);
```

ARBITRAGE DE L'AUDITEUR

REMÉDIATION RECOMMANDÉE

Impact possible : L'effet précis de ce signalement reste à établir : le court extrait ne suffit pas à démontrer un accès indu, une fuite ou une exécution de code.

ACTIONS À EFFECTUER

1. Remplacer MD5/SHA-1 par SHA-256 minimum pour l'intégrité.
2. Pour les mots de passe : bcrypt, scrypt ou Argon2id.
3. Utiliser crypto.randomBytes / random_bytes pour toute valeur de sécurité.

Après — exemple à adapter et tester dans le projet

```
$hash = password_hash($password, PASSWORD_ARGON2ID);
if (password_verify($input, $hash)) { /* ok */ }
```

Conditions à vérifier : Identifier les données d'entrée, le traitement signalé et les protections en amont. Confirmer le comportement dans son contexte avant de qualifier le risque.

Validation après correction : Ajouter un test ciblé reproduisant le cas signalé sans effet destructif, vérifier le résultat attendu et relancer Semgrep. Conserver les preuves de correction et vérifier les autres emplacements du même groupe.

#7 SC2086 — Double Quote To Prevent Globbing And Word Splitting

FAIBLE

outil: ShellCheck règle: shellcheck.SC2086

CWE-78: OS Command Injection A03:2021 - Injection

Variable $TARGET_DIR non entourée de guillemets doubles, risque de découpage de mots si le chemin contient des espaces.

EMPLACEMENTS CONCERNÉS · 1 OCCURRENCE(S), 1 FICHIER(S)

deploy.sh — ligne(s) 14 · 1 occurrence(s)

```
rm -rf $TARGET_DIR/*
```

ARBITRAGE DE L'AUDITEUR

REMÉDIATION RECOMMANDÉE

Impact possible : Un script peut traiter différemment les espaces ou caractères spéciaux, ou ignorer un échec ; l'effet précis dépend de la règle ShellCheck signalée.

ACTIONS À EFFECTUER

1. Consulter la règle SC indiquée et modifier uniquement la construction qu'elle vise, sans appliquer aveuglément des options globales au script.
2. Pour SC2086, citer les expansions comme "$path" ; pour une commande critique, vérifier explicitement son code de retour et traiter l'échec.
3. Ne pas utiliser eval ou source sur des données externes. Tester les changements de comportement avant d'activer set -e ou set -u.

Conditions à vérifier : Vérifier le shell cible et le comportement volontaire des expansions ; certains diagnostics concernent la fiabilité, pas une vulnérabilité confirmée.

Validation après correction : Exécuter shellcheck sur le script puis le tester dans un dossier isolé avec un nom contenant des espaces, une variable vide et une commande en échec. Le résultat doit être correct ou l'échec explicite, sans action hors du dossier de test.

#8 ~~Secret Exposé : Generic API Key (Fixture De Test)~~

INFO

outil: Gitleaks règle: generic-api-key

CWE-798: Use of Hard-coded Credentials

A07:2021 - Identification and Authentication Failures

Clé API générique détectée dans un fichier de simulation de test.

EMPLACEMENTS CONCERNÉS · 1 OCCURRENCE(S), 1 FICHIER(S)

tests/fixtures/auth.ts — ligne(s) 15 · 1 occurrence(s)

```
const MOCK_API_KEY = "sk_test_fake_mock_key_for_testing";
```

ARBITRAGE DE L'AUDITEUR

Auditeur : Faux positif · Info

Clé sk_test dans un fichier de tests unitaires mock. Aucun accès aux flux réels.

Matrice qualitative : **Faible** · Difficile / Faible

2026-11-18 10:32

REMÉDIATION RECOMMANDÉE

Impact possible : Ces fichiers ou contrôles n'ont pas été entièrement examinés ; cela limite les conclusions mais ne démontre pas une faille.

ACTIONS À EFFECTUER

1. Lire le détail de chaque erreur et distinguer parsing, dépendance manquante, délai dépassé et arbitrage de faux positif.
2. Pour Bandit, utiliser une version Python compatible avec le projet et vérifier la liste des fichiers lus. Pour Helm, fournir les valeurs et dépendances requises puis rendre le chart avant le scan.
3. Relancer uniquement les contrôles bloqués et importer les nouvelles sorties ; documenter les parties encore non testées plutôt que supprimer les erreurs.

Conditions à vérifier : Un point reclassé Info peut être un faux positif, pas une erreur d'outil : consulter son arbitrage avant de modifier le code.

Validation après correction : Comparer les erreurs et fichiers analysés avant/après ; les fichiers auparavant bloqués doivent être couverts. Un rapport sans alerte mais sans règles chargées n'est pas un contrôle réussi.

#9 Script Tiers Chargé Sans Attribut Integrity (SRI)

INFO

outil: Semgrep règle: generic.ci.security.missing-sri

CWE-353: Missing Support for Integrity Check

A08:2021 - Software and Data Integrity Failures

Ressource externe chargée depuis un CDN sans empreinte de contrôle d'intégrité (Subresource Integrity).

EMPLACEMENTS CONCERNÉS · 1 OCCURRENCE(S), 1 FICHIER(S)

public/index.html — ligne(s) 12 · 1 occurrence(s)

```
<script src="https://cdn.example.com/lib.js"></script>
```

ARBITRAGE DE L'AUDITEUR

REMÉDIATION RECOMMANDÉE

Impact possible : Ces fichiers ou contrôles n'ont pas été entièrement examinés ; cela limite les conclusions mais ne démontre pas une faille.

ACTIONS À EFFECTUER

1. Lire le détail de chaque erreur et distinguer parsing, dépendance manquante, délai dépassé et arbitrage de faux positif.
2. Pour Bandit, utiliser une version Python compatible avec le projet et vérifier la liste des fichiers lus. Pour Helm, fournir les valeurs et dépendances requises puis rendre le chart avant le scan.
3. Relancer uniquement les contrôles bloqués et importer les nouvelles sorties ; documenter les parties encore non testées plutôt que supprimer les erreurs.

Conditions à vérifier : Un point reclassé Info peut être un faux positif, pas une erreur d'outil : consulter son arbitrage avant de modifier le code.

Validation après correction : Comparer les erreurs et fichiers analysés avant/après ; les fichiers auparavant bloqués doivent être couverts. Un rapport sans alerte mais sans règles chargées n'est pas un contrôle réussi.

05 Méthodologie et confidentialité

- L'audit a été réalisé **intégralement en local**, dans un **environnement cloisonné** (machine WSL dédiée, hors production) : aucun code source n'est téléversé sur un service tiers.
- Les outils (Gitleaks, Semgrep, PHPStan, Trivy…) s'exécutent sur la machine de l'auditeur ; le répertoire temporaire de décompression est **supprimé automatiquement en fin d'exécution**.
- **Aucun secret n'est conservé ni partagé** : les valeurs détectées sont tronquées ou masquées dans le rapport, et rien n'est écrit hors du dossier de destination choisi.
- L'analyse approfondie est facultative : elle ne porte que les constats déjà résumés (règle, message, fichier/ligne, court extrait), jamais les secrets complets ni le code source intégral.
- Les résultats bruts et le rapport restent la propriété du client. **Diffusion restreinte.**
- Cadre de la mission : le client certifie être propriétaire du code fourni ou autorisé à le faire auditer. Il s'agit d'une analyse statique ponctuelle, à une date donnée, qui ne garantit pas l'absence de failles de logique métier ou d'infrastructure.
- Trivy utilise une base de vulnérabilités téléchargée via Internet ; la recherche des dépendances reste locale. Seules les dépendances identifiables avec leur version sont couvertes, Trivy Config vérifie les Dockerfile, docker-compose, Kubernetes et Terraform présents dans l'archive ; les licences et la configuration réellement déployée ne sont pas couvertes.

06 Vérifications manuelles ciblées

Contrôles à réaliser avec autorisation sur un environnement de test. Non exécutés automatiquement. 2 vérifications retenues.

Droits d'accès et accès direct aux objets (IDOR)

Vérification des frontières d'autorisation entre clients.

Tenter d'accéder à l'identifiant de facture d'une autre entreprise depuis une session cliente standard.

Résultat attendu : HTTP 403 Forbidden strict.

État : Conforme

Vérifié : l'accès d'un compte de l'Organisation A aux factures de l'Organisation B renvoie bien HTTP 403 Forbidden.

Invalidation effective de la session après déconnexion

Les jetons JWT stateless restent souvent valides jusqu'à leur expiration naturelle.

Émettre une requête API avec le jeton d'authentification après avoir cliqué sur 'Déconnexion'.

Résultat attendu : HTTP 401 Unauthorized dès la déconnexion.

État : Échec

Point d'attention : le jeton JWT reste accepté par l'API pendant 15 minutes après l'appel à /api/logout (manque d'invalidation serveur).