AYI NEDJIMI Consultants

Détection de secrets dans le code : Gitleaks et CI/CD

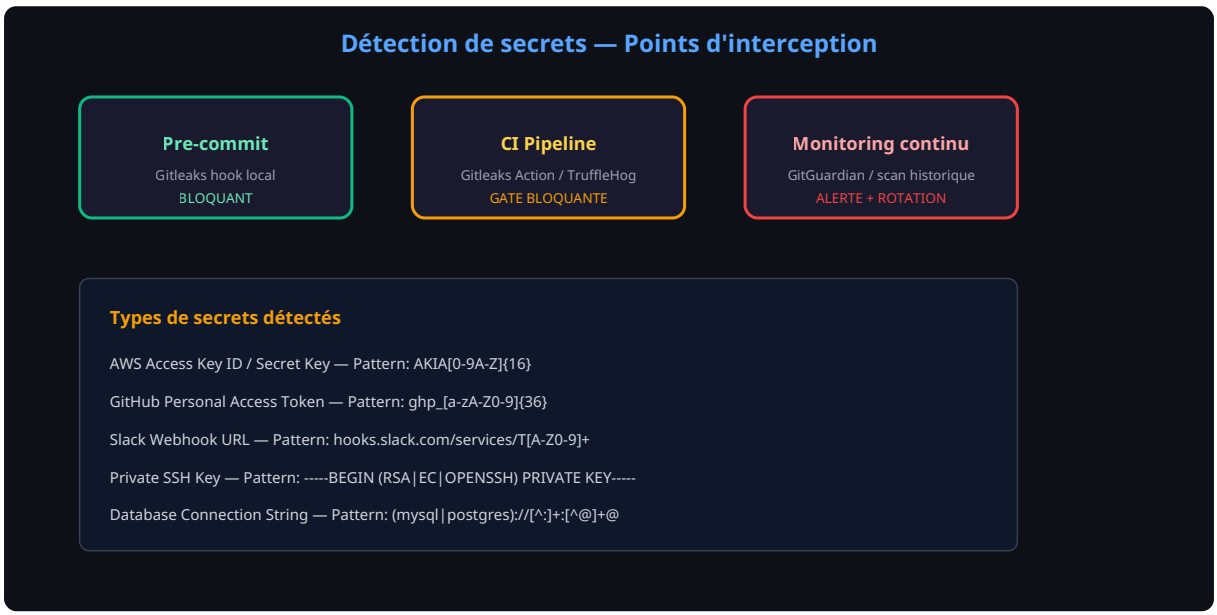
Catégorie : DevSecOps Lecture : 4 min Publié le : 12/03/2026 Auteur : Ayi NEDJIMI

Détectez les secrets exposés dans votre code avec Gitleaks, TruffleHog et les pre-commit hooks. Guide pratique d'intégration CI/CD et remédiation.

Un développeur commit un fichier .env contenant une clé API AWS avec des droits administrateur. Vingt minutes plus tard, un bot a déjà scanné le dépôt public, extrait la clé et lancé des instances EC2 pour du cryptomining. Ce scénario se produit des milliers de fois par semaine sur GitHub. Selon les données de GitGuardian, plus de 12 millions de secrets ont été exposés sur des dépôts publics en 2024. Mais le problème ne concerne pas que les projets open-source : vos dépôts privés contiennent probablement des secrets qui n'ont rien à y faire — tokens d'API, clés SSH, mots de passe de bases de données, certificats TLS. La détection de secrets dans le code est devenue une brique indispensable de toute stratégie DevSecOps. Gitleaks, TruffleHog et les pre-commit hooks forment la première ligne de défense. Ce guide vous montre comment déployer ces outils efficacement, les intégrer dans votre pipeline CI/CD et gérer les alertes sans noyer vos équipes sous les faux positifs.

Points clés à retenir

- **Gitleaks** en pre-commit hook intercepte les secrets avant qu'ils n'atteignent le dépôt distant — c'est la protection la plus efficace
- **TruffleHog** excelle pour le scan de l'historique Git et la détection de secrets dans les commits passés
- La rotation immédiate du secret est la seule réponse valable — révoquer et remplacer, jamais seulement supprimer du code
- Un **allowlist** bien maintenue réduit les faux positifs de 70% sans compromettre la détection



Gitleaks : la référence pour la détection de secrets

Gitleaks est l'outil open-source de détection de secrets le plus utilisé dans l'écosystème DevSecOps. Écrit en Go, il est rapide (scanne un dépôt de 100K commits en moins de 2 minutes), fiable, et maintient un ensemble de 150+ règles de détection couvrant les principaux fournisseurs cloud et services SaaS.

```
# Scan du répertoire courant
gitleaks detect -v

# Scan de l'historique Git complet
gitleaks detect --source . --log-opts="--all" -v

# Scan uniquement des nouveaux commits (CI)
gitleaks detect --log-opts="origin/main..HEAD" -v

# Avec un fichier de configuration personnalisé
gitleaks detect -c .gitleaks.toml -v
```

La force de Gitleaks réside dans sa configurabilité. Le fichier `.gitleaks.toml` permet d'ajouter des règles custom, de définir des allowlists par chemin ou par pattern, et de personnaliser la sévérité. C'est un outil que vous pouvez adapter précisément à votre contexte sans modifier le code source. Pour la gestion centralisée des secrets détectés, consultez notre guide sur le secrets management cloud.

TruffleHog : spécialiste de l'historique Git

TruffleHog (de Truffle Security) excelle dans un domaine spécifique : le scan de l'historique Git en profondeur. Là où Gitleaks scanne les fichiers actuels et les diffs, TruffleHog analyse chaque commit, chaque branche, chaque tag. Il détecte les secrets qui ont été commités puis supprimés — mais qui restent dans l'historique Git.

TruffleHog v3 introduit une fonctionnalité unique : la **vérification active**. Pour chaque secret trouvé, l'outil tente de l'utiliser (appel API avec la clé) pour confirmer s'il est encore valide. Pas de faux positif : si le secret fonctionne, l'alerte est confirmée. Cette approche est particulièrement utile pour nettoyer un dépôt avec un long historique.

```
# Scan complet avec vérification active
trufflehog git file://. --only-verified

# Scan d'un dépôt GitHub distant
trufflehog github --repo https://github.com/org/repo --only-verified
```

Attention : la vérification active génère du trafic vers les APIs des fournisseurs. Utilisez-la uniquement sur vos propres dépôts et en dehors des environnements de production. C'est un outil d'audit, pas un outil de CI quotidien. Pour l'analyse de la prolifération de secrets à grande échelle, notre article sur le secrets sprawl complète cette approche.

Pre-commit hooks : la première ligne de défense

Le meilleur moment pour détecter un secret, c'est avant qu'il n'entre dans l'historique Git. Les **pre-commit hooks** interceptent le commit localement, sur la machine du développeur, et bloquent l'opération si un secret est détecté.

```
# .pre-commit-config.yaml
repos:
  - repo: https://github.com/gitleaks/gitleaks
    rev: v8.18.0
    hooks:
      - id: gitleaks
```

Installez le framework `pre-commit` et distribuez la configuration à toute l'équipe. Le scan en pre-commit ne prend que quelques secondes car il ne vérifie que les fichiers staged, pas tout le dépôt. C'est transparent pour le développeur dans 99% des cas.

Le point faible : les hooks sont côté client et peuvent être contournés avec `--no-verify`. C'est pourquoi le scan en CI est indispensable comme filet de sécurité. Le pre-commit est une courtoisie envers le développeur (feedback immédiat), le CI est la gate bloquante réelle.

Remédiation : rotation immédiate et nettoyage

Quand un secret est détecté dans un commit, la seule réponse valable est la **rotation** : révoquez l'ancien secret et générez-en un nouveau. Supprimer le fichier dans un commit ultérieur ne sert à rien — l'historique Git conserve l'ancienne version indéfiniment. Un attaquant peut toujours extraire le secret avec `git log -p`.

Procédure de remédiation en 4 étapes :

1. **Révoquer** — Désactivez immédiatement le secret compromis dans le service concerné (console AWS, dashboard Stripe, etc.).

2. **Évaluer l'impact** — Vérifiez les logs d'accès pour déterminer si le secret a été utilisé par un tiers.
3. **Remplacer** — Générez un nouveau secret et stockez-le dans votre vault (HashiCorp Vault, AWS Secrets Manager, Azure Key Vault).
4. **Nettoyer l'historique** — Si nécessaire, utilisez `git filter-branch` ou BFG Repo-Cleaner pour purger l'historique.

Selon les données de GitGuardian State of Secrets Sprawl 2025, le délai moyen de remédiation est de 27 jours. Cible réaliste pour une organisation mature : moins de 4 heures pour les secrets critiques (clés cloud, tokens d'accès admin). La mise en place d'un playbook de réponse aux incidents accélère considérablement ce processus.

Réduire les faux positifs avec une allowlist intelligente

Un outil de détection de secrets qui produit trop de faux positifs finit ignoré. La clé : une **allowlist** bien maintenue dans votre fichier `.gitleaks.toml` :

```
[allowlist]
description = "Exceptions documentées"
paths = [
  "tests/fixtures/",
  "docs/examples/",
]
regexes = [
  "EXAMPLE_KEY_DO_NOT_USE",
  "test-api-key-[a-z]+",
]
```

Autorisez les chemins de tests et les clés d'exemple documentées. Mais chaque entrée dans l'allowlist doit être justifiée et revue régulièrement. Selon mon expérience, une allowlist de 15-20 règles couvre 90% des faux positifs sans compromettre la détection. Le rapport NIST Software Supply Chain Security Guidance recommande cette approche de gestion des exceptions documentées.

Sources et références : OWASP DevSecOps · NIST

Questions fréquentes sur la détection de secrets

Gitleaks ou TruffleHog, lequel choisir ?

Les deux se complètent. Gitleaks est idéal pour le CI/CD quotidien (rapide, configurable, peu de faux positifs). TruffleHog est parfait pour les audits ponctuels de dépôts existants grâce à sa vérification active. Si vous ne devez en choisir qu'un pour la CI, prenez Gitleaks. Si vous devez auditer un vieux dépôt, utilisez TruffleHog.

Comment gérer un secret déjà pushé sur un dépôt public ?

Considérez-le comme compromis, même si vous l'avez supprimé 30 secondes après. Les bots scannent GitHub en continu. Révoquez le secret immédiatement, évaluez l'impact (logs CloudTrail pour AWS, audit logs du service), générez un remplacement et stockez-le dans un vault. Le nettoyage de l'historique Git est souhaitable mais ne remplace pas la rotation.

Peut-on utiliser ces outils sur des dépôts très volumineux ?

Oui. Gitleaks scanne un dépôt de 500K commits en 5-10 minutes. TruffleHog est plus lent sur l'historique complet mais propose un mode --since-commit pour limiter le scope. Pour les mono-repos très volumineux, segmentez le scan par répertoire dans votre pipeline CI.

Ayi NEDJIMI Consultants — Expert cybersécurité offensive & intelligence artificielle

ayinedjimi-consultants.fr · ayi@ayinedjimi-consultants.fr