FR

Guide Pratique

OpenClaw : Secrets dans le Contexte IA

Credentials, Cles API et Vault

Vault, injection dynamique et protection des secrets pour agents IA

Mars 2026

Série OpenClaw — Billet 4/7

Ne jamais exposer ses secrets dans le contexte IA

Credentials, clés API et vault — architecture BOTUM complète

1. Le paradoxe fondamental

Pour qu'un agent OpenClaw soit réellement autonome, il doit avoir accès aux services sur lesquels il agit : clé API GitHub, token SMTP, mot de passe Vaultwarden, accès SSH. Le paradoxe : l'agent a besoin de ces secrets pour opérer — mais la façon dont on les lui fournit est critique.

Passer une clé API directement dans un message, la coder en dur dans un script du workspace, ou la déposer dans MEMORY.md revient à l'exposer dans le contexte IA — une surface d'attaque à part entière.

2. Pourquoi le contexte IA est une surface d'attaque

Les logs d'historique

OpenClaw conserve l'historique des échanges. Telegram garde l'historique des messages. Si une clé API transite dans un message ou dans un fichier de workspace versionné, elle persiste dans des endroits auxquels on ne pense pas. Un git log devient un vecteur d'exfiltration.

Les réponses du modèle

Les LLMs peuvent reproduire des données qu'ils ont vues dans leur contexte. Plusieurs incidents documentés impliquent des modèles "citant" des tokens de sécurité qui traînaient dans leur contexte d'entrée.

La prompt injection

Si un agent traite du contenu externe (emails, pages web, issues GitHub), un acteur malveillant peut injecter des instructions pour exfiltrer les secrets présents dans le contexte. Vecteur documenté par OWASP LLM Security Top 10.

La persistance mémoire

MEMORY.md, CODEX.md, les fichiers de state — tout ce qui persiste dans le workspace peut être relu par l'agent dans des sessions futures, ou par d'autres agents du réseau.

3. Les erreurs classiques

La clé API dans MEMORY.md

L'agent note la clé SMTP dans MEMORY.md pour ne pas redemander à chaque session. Résultat : la clé est dans le contexte permanent de toutes les sessions et dans Git.

Le mot de passe dans un message Telegram

Mot de passe envoyé par Telegram "pour aller vite". Stocké côté Telegram (serveurs tiers), dans l'historique OpenClaw, et potentiellement loggé localement — au minimum 3 endroits non contrôlés.

Le hardcode dans un script du workspace

API_KEY = "sk-..." en dur dans un script Python. Le workspace est Git. Un git log -p retrouvera la clé même après suppression.

Les credentials dans les fichiers de config

openclaw.json ou config.yaml avec tokens en clair. Un commit accidentel ou un partage de fichier = fuite garantie.

4. La solution : vault + injection dynamique

Règle d'or : les secrets ne vivent que dans le vault, et ne sont récupérés qu'au moment de l'exécution. Jamais en dur, jamais dans le contexte, jamais dans un fichier versionné.

Pattern recommandé (shell)

```
# Bon pattern : récupération au moment de l'exécution TOKEN=$(bw get password
"api-service-x") curl -H "Authorization: Bearer $TOKEN" https://api.service-x.com/ unset
TOKEN # nettoyage immédiat
```

Pattern à éviter absolument

```
# MAUVAIS PATTERN — Ne jamais faire TOKEN="sk-abc123xyz" # hardcode = risque permanent curl
-H "Authorization: Bearer $TOKEN" https://api.service-x.com/
```

Avec le vault comme source unique de vérité, la rotation d'un secret est triviale : on change la valeur dans Vaultwarden, tous les scripts récupèrent automatiquement la nouvelle valeur.

5. Variables d'env vs Vault — quand utiliser quoi

Cas d'usage	Variables env	Vault
Secrets d'infrastructure (Docker, systemd)	✓ Adapté	Possible
Secrets partagés entre plusieurs agents	✗ Risqué	✓ Recommandé
Credentials rotatifs fréquemment	✗ Lourd	✓ Idéal
Secrets injectés dans le contexte IA	✗ À éviter	✗ À éviter
Tokens éphémères (OAuth, JWT)	✓ Adapté	Possible

Règle finale : aucun secret ne doit jamais être injecté dans le contexte de l'agent IA — ni via variable d'environnement, ni via vault. L'agent invoque un outil qui récupère et utilise le secret en isolation, sans que ce secret transite dans la fenêtre de contexte du LLM.

6. Bonnes pratiques OpenClaw

Le workspace Git : code + config, jamais de secrets

Le workspace OpenClaw est versionné par Git. Tout ce qui est commité devient permanent (même après suppression, le git log garde une trace).

Ce qui peut être commité : code, scripts (sans credentials), fichiers de configuration sans valeurs sensibles, templates. Ce qui ne doit jamais être commité : clés API, mots de passe, tokens OAuth, certificats privés, clés SSH privées.

.gitignore strict — liste minimale

```
# Secrets — JAMAIS dans Git .env | .env.* | *.key | *.pem | *.p12 secrets.json |
config.local.yaml | credentials.json # OpenClaw spécifiques MEMORY.md | openclaw.json |
*.session
```

Rotation régulière des secrets

Selon l'étude CyberArk 2025, 68% des incidents de sécurité impliquant des credentials sont liés à des secrets qui n'ont jamais été rotés. Recommandation BOTUM : 90 jours pour les tokens d'infrastructure, 30 jours pour les clés d'accès API à haute valeur.

7. Scénario réel : l'architecture BOTUM

Couche 1 : Vaultwarden self-hosted

Un Vaultwarden tourne sur us-srv-dck-01, accessible uniquement depuis le réseau interne. Collections : infrastructure, apis-externes, agents-openclaw, smtp.

Couche 2 : Script d'injection isolé

Chaque agent dispose d'un helper script vault-inject.sh qui se connecte au vault via bw CLI avec un token éphémère, récupère le secret, l'injecte en variable temporaire, exécute l'action, et nettoie immédiatement.

Couche 3 : Aucune persistance dans le contexte

Les agents ont accès au nom des secrets dans le vault ("github-token-fdbot"), pas aux secrets eux-mêmes. Le secret ne transite jamais dans la fenêtre de contexte LLM.

Couche 4 : Audit et alertes

KNOX, l'agent sécurité, génère un rapport hebdomadaire sur l'usage des credentials : accès par agent, détection d'anomalies, rappels de rotation.

8. Checklist — 10 règles pour ne jamais leaker un secret

1. Tout secret vit dans le vault — Vaultwarden, HashiCorp Vault, ou équivalent. Nulle part ailleurs.

2. Zéro secret dans MEMORY.md, CODEX.md, ou tout fichier workspace — persistants et lus par tous les agents.

3. Zéro secret dans les messages Telegram/Slack/email — les messageries ne sont pas des gestionnaires de secrets.

4. Zéro hardcode dans les scripts — même en dev local, même "temporairement".

5. .gitignore exhaustif — inclure tous les patterns de fichiers sensibles avant le premier commit.

6. Injection dynamique uniquement — le secret est récupéré et utilisé en mémoire, jamais écrit.

7. Principe du moindre privilège — chaque agent n'accède qu'aux secrets dont il a strictement besoin.

8. Rotation sur calendrier — 90 jours infra, 30 jours APIs haute valeur.

9. Audit régulier des accès — qui a accédé à quoi, quand, depuis quel agent.

10. Clean up des secrets révoqués — un token expiré est supprimé du vault immédiatement.

Conclusion

La gestion des secrets dans un réseau d'agents IA n'est pas un problème marginal. C'est l'un des vecteurs de risque les plus silencieux de l'automatisation moderne. L'approche vault central + injection dynamique + aucun secret dans le contexte IA + rotation sur calendrier élimine une catégorie entière de risques de façon durable.

→ Billet 5 : Configurer ses premiers agents spécialisés — JARVIS, HERMÈS, CHRONOS