Chapitre C — Sécurité & Audit

2 skills regroupés sous ce thème.

- cso
- guard

cso

cso

> *Catalogue généré le 2026-05-11*

En une phrase

Mode « Chief Security Officer » : Claude audite ton projet à la recherche de failles de sécurité, secrets exposés, dépendances vulnérables et risques d'attaque.

Quand l'utiliser

- Avant de déployer une application sensible (qui touche à de l'argent, des données personnelles, de l'auth)
- Quand tu veux vérifier qu'aucun mot de passe ou clé API ne traîne dans ton historique git
- Quand tu veux un audit OWASP Top 10 (les dix grandes catégories de vulnérabilités web)
- Après avoir ajouté de nouvelles dépendances, pour vérifier qu'elles ne contiennent pas de failles connues
- Périodiquement (par exemple chaque mois) pour suivre l'évolution de la sécurité de ton projet

Comment l'invoquer

- **Slash command** : `/cso` (à taper dans Claude Code)
- **Voice triggers** : « see-so » · « see so » · « security review » · « security check » · « vulnerability scan » · « run security »
- **Phrases déclencheurs (texte)** : "security audit" / "threat model" / "pentest review" / "owasp review" / "CSO review"
- **Auto-invocation** : Sur demande explicite

Description détaillée

`/cso` lance un audit de sécurité à la fois sur ton code et sur ton infrastructure. Le skill détecte d'abord ton stack technique (Node, Python, Ruby, Go, etc.) puis cartographie la surface d'attaque : endpoints publics, routes authentifiées, points d'upload de fichiers, intégrations externes, webhooks, jobs en arrière-plan.

Il enchaîne plusieurs phases. **Archéologie des secrets** : il fouille l'historique git pour repérer les clés AWS (`AKIA…`), clés OpenAI (`sk-…`), tokens GitHub, etc. qui auraient pu être commit par erreur. **Chaîne de dépendances** : il lance `npm audit` ou équivalent, repère les scripts d'install suspects dans tes dépendances de production. **Pipeline CI/CD** : il vérifie tes workflows GitHub Actions (actions non pinnées, `pull_request_target` dangereux, injection de scripts). **OWASP Top 10 + STRIDE** : injections SQL, XSS, CSRF, contrôles d'accès cassés, etc.

Deux modes existent. **Daily** est silencieux (signale seulement à 8/10 de confiance, zéro bruit). **Comprehensive** est l'audit mensuel approfondi (à 2/10, tout sort). Tu peux aussi cibler des sous-périmètres : `--infra`, `--code`, `--skills`, `--diff` (limité aux changements de ta branche). À chaque exécution, les résultats sont stockés pour suivre les tendances dans le temps.

Source

- **Plugin** : `gstack`
- **Nom interne** : `cso`
- **Fichier** : `/home/thymon/.claude/skills/gstack/cso/SKILL.md`

guard

guard

> *Catalogue généré le 2026-05-11*

En une phrase

Mode « sécurité maximale » : Claude t'avertit avant chaque commande destructrice ET t'empêche de modifier des fichiers hors d'un dossier précis.

Quand l'utiliser

- Quand tu travailles sur de la production en direct et tu veux zéro accident
- Quand tu débugges un système critique et tu veux limiter le périmètre des modifications
- Quand tu donnes la main à Claude sur un projet sensible et tu veux des garde-fous serrés
- Quand tu veux la combinaison `/careful` + `/freeze` activée en une seule commande
- Avant de toucher à n'importe quoi qui pourrait coûter cher si ça casse

Comment l'invoquer

- **Slash command** : `/guard` (à taper dans Claude Code)
- **Phrases déclencheurs (texte)** : "guard mode" / "full safety" / "lock it down" / "maximum safety"
- **Auto-invocation** : Sur demande explicite

Description détaillée

`/guard` active deux protections d'un coup. La première vient de `/careful` : avant d'exécuter une commande potentiellement destructrice (`rm -rf`, `DROP TABLE`, `git push --force`, `git reset --hard`, `kubectl delete`, `docker system prune`, etc.), Claude s'arrête et te demande confirmation. Tu peux toujours autoriser, mais tu ne peux pas le faire sans le voir. Certains cas évidents (`rm -rf node_modules`, `rm -rf .next`, etc.) restent autorisés sans alerte.

La seconde vient de `/freeze` : tu choisis un dossier précis (par exemple `src/auth/`) et toutes les modifications de fichiers en dehors de ce dossier sont bloquées. Quand le skill démarre, il te demande où placer cette frontière, puis enregistre le dossier dans son fichier d'état.

Les deux protections fonctionnent via des « hooks » Claude Code qui interceptent chaque appel aux outils Bash, Edit et Write avant exécution. Tu vois passer un statut « Checking for destructive commands… » ou « Checking freeze boundary… » à chaque action sensible. Pour relâcher la contrainte sur le dossier, tu lances `/unfreeze`. Pour tout désactiver, tu termines la session — les hooks sont session-scopés.

C'est le mode à activer quand tu veux que Claude soit utile mais qu'il ne puisse pas tout casser par excès de zèle.

Source

- **Plugin** : `gstack`
- **Nom interne** : `guard`
- **Fichier** : `/home/thymon/.claude/skills/gstack/guard/SKILL.md`