variant-analysis

variant-analysis

> *Catalogue généré le 2026-05-11*

En une phrase

Quand on a trouvé un bug à un endroit, ce skill cherche systématiquement le même type de bug ailleurs dans le code — parce qu'une erreur faite une fois est souvent reproduite plusieurs fois.

Quand l'utiliser

- Tu viens de découvrir une faille à un endroit et tu veux savoir si elle est répétée ailleurs.
- Tu veux construire une règle automatique (avec Semgrep ou CodeQL, deux outils qui scannent le code) pour repérer un motif vulnérable.
- Après un audit qui a trouvé un problème : tu veux remonter à la racine et débusquer tous les cousins.
- Quand un correctif vient d'être publié pour une bibliothèque et que tu veux vérifier si ton code contient le même schéma vulnérable.

Comment l'invoquer

- **Slash command** : `/variant-analysis` (ou `/variants`).
- **Phrases déclencheurs (texte)** : "find similar bugs", "hunt variants", "build a CodeQL query", "search for this pattern".
- **Auto-invocation** : Sur demande explicite.

Description détaillée

Le principe de base de ce skill, c'est qu'un bug est rarement seul. Quand un développeur fait une erreur dans une partie du code (par exemple : oublier de vérifier qu'une valeur n'est pas négative), il a probablement fait la même erreur à plusieurs autres endroits, par copier-coller ou simplement par habitude. Plutôt que d'attendre que ces bugs frères soient découverts un par un, on les chasse tous d'un coup.

Le skill fonctionne en cinq étapes : 1) bien comprendre la cause racine du bug initial (pas juste son symptôme), 2) écrire un motif de recherche qui retrouve exactement le bug d'origine, 3) repérer les éléments qu'on peut généraliser (le nom de la variable n'a pas d'importance, mais la structure de l'appel oui), 4) élargir le motif petit à petit en vérifiant qu'on ne génère pas trop de faux positifs (alertes injustifiées), 5) trier les résultats par exploitabilité.

En entrée, on lui donne un bug connu. En sortie, on obtient une liste de candidats potentiels dans tout le projet, classés par niveau de confiance (élevé / moyen / faible), avec leur fichier et leur ligne. Il choisit son outil selon le besoin : `ripgrep` pour une recherche textuelle rapide, Semgrep pour des motifs syntaxiques, ou CodeQL pour suivre des flux de données complexes à travers le code.

Pour aller plus loin

Pour les détails techniques, exemples et patterns spécifiques, voir le SKILL.md original.

Source

- **Plugin** : `trailofbits/variant-analysis`
- **Nom interne** : `variant-analysis`
- **Fichier** : `/home/thymon/.claude/plugins/cache/trailofbits/variant-analysis/1.0.0/skills/variant-analysis/SKILL.md`

Revision #2
Created 2026-05-11 21:19:25 UTC by thymon
Updated 2026-05-11 21:37:03 UTC by thymon