constant-time-analysis

constant-time-analysis

> *Catalogue généré le 2026-05-11*

En une phrase

Détecte les bouts de code cryptographique qui mettent un peu plus (ou un peu moins) de temps à s'exécuter selon la valeur d'un secret — ce qui peut permettre à un attaquant de deviner ce secret juste en mesurant les temps de réponse.

Quand l'utiliser

- Tu écris ou tu relis du code qui manipule des clés cryptographiques, des mots de passe, des tokens d'authentification.
- Tu utilises des opérations sensibles comme la division `/` ou le modulo `%` sur des valeurs secrètes.
- Tu touches à une fonction `sign`, `verify`, `encrypt`, `decrypt` ou `derive_key`.
- Tu as entendu parler de « timing attack » ou « side-channel » et tu veux savoir si ton code est concerné.

Comment l'invoquer

- **Slash command** : `/constant-time-analysis` (ou `/ct-check`).
- **Phrases déclencheurs (texte)** : "constant-time", "timing attack", "side-channel", "KyberSlash".
- **Auto-invocation** : Sur demande explicite, ou quand tu écris du code crypto.

Description détaillée

Imagine un cadenas digital qui dit « bipe court » quand le premier chiffre est faux et « bipe long » quand il est juste. Tu n'as plus qu'à essayer de 0 à 9 sur le premier chiffre, retenir lequel a fait le bip long, puis passer au deuxième. En quelques secondes, tu ouvres un cadenas censé être inviolable. C'est exactement le principe des « attaques par canal auxiliaire par mesure de temps » (timing side-channel) : un attaquant ne voit pas le secret, mais il voit combien de temps le programme met à répondre, et ça lui suffit parfois à le déduire.

Pour qu'un code cryptographique soit sûr, il doit prendre **exactement le même temps** quel que soit le secret manipulé — on dit qu'il est « à temps constant ». Concrètement, ça veut dire éviter les branchements `if (secret == x)` qui prennent un chemin différent selon la valeur, et éviter certaines opérations comme la division qui peuvent être plus rapides sur certains nombres que sur d'autres. Le skill analyse le code (en C, C++, Go, Rust, Java, Python, JavaScript et plein d'autres langages) et signale tous les endroits qui ne respectent pas cette contrainte.

Il fournit un script `ct_analyzer` qui produit un rapport détaillé : chaque opération suspecte, sa ligne dans le code, et pourquoi elle est risquée. Il peut sortir en JSON pour intégration dans un pipeline CI. Particulièrement utile sur les opérations sensibles modernes (signatures Ed25519, encapsulation post-quantique type Kyber, etc.). À noter : ce skill n'est pas pertinent pour du code « métier » classique — il vise uniquement le code qui manipule des secrets cryptographiques.

Pour aller plus loin

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

Source

- **Plugin** : `trailofbits/constant-time-analysis`
- **Nom interne** : `constant-time-analysis`
- **Fichier** : `/home/thymon/.claude/plugins/cache/trailofbits/constant-time-analysis/0.1.0/skills/constant-time-analysis/SKILL.md`

Revision #2
Created 2026-05-11 21:19:59 UTC by thymon
Updated 2026-05-11 21:37:32 UTC by thymon