substrate-vulnerability-scanner

substrate-vulnerability-scanner

> *Catalogue généré le 2026-05-11*

En une phrase

Scanne le code d'une blockchain construite avec Substrate (la techno derrière Polkadot et ses "parachains") pour repérer 7 types de failles critiques qui peuvent faire crasher un nœud, ouvrir des attaques de spam ou bypasser l'admin.

Quand l'utiliser

- Quand un projet utilise Substrate ou FRAME (le framework pour fabriquer une blockchain "à la Polkadot") et que tu veux faire un check sécurité.
- Avant de lancer une parachain Polkadot/Kusama sur le réseau principal.
- Quand un dev a écrit un "pallet" custom (un module qui ajoute des fonctionnalités à la chaîne) et que tu veux le faire relire.
- Pour vérifier que les calculs de "poids" (le coût d'une transaction) sont corrects — un mauvais poids ouvre des attaques de spam.
- Si tu auditeras des fonctions privilégiées (qui ne devraient être appelables que par root/admin).

Comment l'invoquer

- **Slash command** : `/substrate-vulnerability-scanner`
- **Phrases déclencheurs (texte)** : "audit my Substrate pallet" / "check FRAME runtime" / "scan Polkadot parachain"
- **Auto-invocation** : Sur demande explicite (généralement lors d'un audit).

Description détaillée

Substrate est le framework Rust de Parity pour construire des blockchains personnalisées (notamment les parachains Polkadot). Le code est organisé en "pallets" (modules) qui sont assemblés dans un runtime. Les bugs Substrate sont souvent catastrophiques : un panic dans un pallet peut faire planter tous les nœuds du réseau, et une mauvaise vérification d'origine peut donner les droits admin à n'importe qui.

Ce skill cherche 7 patterns critiques : 1) **Arithmetic Overflow** (utiliser `+`, `-`, `*` directement au lieu de `checked_*` — les calculs débordent silencieusement en mode release), 2) **Don't Panic** (un `unwrap()` qui plante = nœud HS = blockchain HS), 3) **Weights & Fees mal calibrés** (ouvre des attaques DoS), 4) **Verify First, Write Last** (écrire en storage avant de valider — sur d'anciennes versions, l'écriture persiste même si la validation échoue), 5) **Unsigned Transaction Validation** (transactions sans signataire mal protégées contre le replay), 6) **Bad Randomness** (utiliser un random prédictible/manipulable), 7) **Bad Origin** (utiliser `ensure_signed` au lieu de `ensure_root` sur une fonction admin — n'importe qui peut l'appeler).

Workflow : tu pointes ton dossier `pallets/`, le skill scanne, te liste les findings avec sévérité et te propose les fix Rust.

Pour aller plus loin

Pour les détails techniques (patterns exacts, code Rust avant/après, intégration `cargo-fuzz` et `try-runtime`), consulter le SKILL.md original à `/home/thymon/.claude/plugins/cache/trailofbits/building-secure-contracts/1.0.1/skills/substrate-vulnerability-scanner/SKILL.md` et la ressource `resources/VULNERABILITY_PATTERNS.md`.

Source

- **Plugin** : `trailofbits/building-secure-contracts`
- **Nom interne** : `substrate-vulnerability-scanner`

- **Fichier** : `/home/thymon/.claude/plugins/cache/trailofbits/building-secure-contracts/1.0.1/skills/substrate-vulnerability-scanner/SKILL.md`

Revision #2
Created 2026-05-11 21:19:31 UTC by thymon
Updated 2026-05-11 21:37:08 UTC by thymon