solana-vulnerability-scanner

solana-vulnerability-scanner

> *Catalogue généré le 2026-05-11*

En une phrase

Scanne automatiquement le code d'un programme Solana pour repérer 6 types de failles de sécurité critiques propres à cette blockchain (validation des comptes, signataires, adresses dérivées, appels inter-programmes, etc.).

Quand l'utiliser

- Avant de déployer un programme Solana (en Rust natif ou avec le framework Anchor) sur le réseau principal.
- Quand tu veux faire auditer un projet Solana ou préparer son audit externe.
- Quand tu intègres un programme qui appelle d'autres programmes ("CPI", c'est-à-dire un programme qui en invoque un autre) et que tu veux vérifier que c'est fait proprement.
- Pour valider la gestion des PDA (Program Derived Addresses, des adresses "calculées" à partir d'une formule et utilisées comme comptes spéciaux).
- Quand un dev t'envoie son code Solana et que tu veux faire un premier filtre rapide.

Comment l'invoquer

- **Slash command** : `/solana-vulnerability-scanner`
- **Phrases déclencheurs (texte)** : "audit my Solana program" / "check this Anchor contract" / "scan Solana CPI"
- **Auto-invocation** : Sur demande explicite (généralement lors d'un audit).

Description détaillée

Solana est une blockchain rapide où les programmes (l'équivalent des smart contracts) sont écrits en Rust, parfois avec un framework appelé Anchor. Le modèle Solana est très particulier : chaque transaction passe une liste de "comptes" au programme, et c'est au programme de vérifier que ces comptes sont les bons. Beaucoup de hacks viennent d'un oubli de vérification.

Ce skill cherche 6 catégories de bugs classiques : 1) **Arbitrary CPI** (le programme appelle un autre programme sans vérifier que c'est le bon — un attaquant peut substituer un programme malveillant), 2) **PDA mal validée** (l'adresse dérivée n'est pas calculée canoniquement, donc usurpable), 3) **Owner check manquant** (on lit les données d'un compte sans vérifier qui le possède), 4) **Signer check manquant** (on autorise une action sans vérifier que la bonne personne a signé), 5) **Sysvar usurpé** (utilisation d'anciennes fonctions non sécurisées pour lire les variables système), 6) **Mauvaise introspection des instructions**.

Le workflow : tu lui donnes ton code Rust/Anchor, il fait des recherches `ripgrep` ciblées, te liste les findings avec gravité (CRITICAL/HIGH/MEDIUM), pointe les lignes concernées et propose un patch.

Pour aller plus loin

Pour les détails techniques (patterns exacts, exemples de code vulnérable vs corrigé, tests Anchor pour reproduire les attaques), consulter le SKILL.md original à `/home/thymon/.claude/plugins/cache/trailofbits/building-secure-contracts/1.0.1/skills/solana-vulnerability-scanner/SKILL.md` et la ressource `resources/VULNERABILITY_PATTERNS.md`.

Source

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

Revision #2
Created 2026-05-11 21:19:37 UTC by thymon
Updated 2026-05-11 21:37:13 UTC by thymon