dwarf-expert

dwarf-expert

> *Catalogue généré le 2026-05-11*

En une phrase

Apporte une expertise pointue sur DWARF — un format technique caché dans les programmes compilés qui permet aux outils de debug (comme `gdb`) de savoir à quelle ligne de code source correspond chaque instruction machine.

Quand l'utiliser

- Tu travailles sur un programme compilé en C, C++, Rust ou Go et tu veux comprendre comment ses informations de debug sont stockées.
- Tu utilises des outils comme `dwarfdump` ou `readelf` pour examiner un binaire (un fichier exécutable).
- Tu écris ou tu relis du code qui lit/produit des données DWARF (par exemple : un débogueur, un profileur, un outil de couverture de code).
- Tu veux valider qu'un binaire contient bien des infos de debug exploitables (`llvm-dwarfdump --verify`).

Comment l'invoquer

- **Slash command** : `/dwarf-expert`.
- **Phrases déclencheurs (texte)** : "DWARF debug info", "parse DWARF", "dwarfdump", "readelf", "debug symbols".

- **Auto-invocation** : Sur demande explicite, ou quand tu mentionnes DWARF dans une question.

Description détaillée

Quand on compile un programme (en C, C++, Rust…), le code source lisible est transformé en code machine illisible. Mais pour pouvoir déboguer ce programme plus tard, le compilateur joint un dictionnaire à part qui dit : « cette suite d'instructions machine correspond à la ligne 42 du fichier `main.c`, la variable `x` est stockée dans le registre EAX à ce moment-là, etc. ». Ce dictionnaire suit un standard appelé DWARF (versions 3, 4 et 5). C'est lui qui permet à `gdb` ou à un profileur de te montrer du code source plutôt que de l'assembleur.

Ce skill apporte la connaissance technique pour manipuler ces données : comment elles sont structurées, comment les extraire avec les outils standards (`dwarfdump`, `readelf`, `llvm-dwarfdump`), comment les générer correctement, et comment écrire du code qui les analyse à l'aide de bibliothèques comme `libdwarf`, `pyelftools` ou `gimli` (côté Rust).

Il est utile dans plusieurs scénarios concrets : développer un outil de couverture de code, analyser un binaire suspect dans un contexte de reverse engineering, vérifier qu'un binaire de production est correctement compilé avec ses infos de debug, ou comparer la qualité du debug entre deux versions d'un compilateur. C'est un skill très spécialisé, qui ne sert que si tu touches à des binaires compilés natifs (pas du JavaScript, pas du Python).

Pour aller plus loin

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

Source

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

Revision #2
Created 2026-05-11 21:19:42 UTC by thymon
Updated 2026-05-11 21:37:18 UTC by thymon