harness-writing

harness-writing

> *Catalogue généré le 2026-05-11*

En une phrase

Le harness (ou "harnais") c'est la petite fonction d'entrée qu'on écrit pour brancher un fuzzer à son code : un mauvais harness = aucun bug trouvé, donc c'est l'élément le plus critique d'une campagne de fuzzing.

Quand l'utiliser

- Démarrer une nouvelle campagne de fuzzing (avant tout, écrire le bon harness).
- Améliorer un harness existant qui ne trouve plus de bugs.
- Convertir des bytes aléatoires en arguments typés pour ta fonction (FuzzedDataProvider).
- Garantir que ton harness est déterministe (même entrée = même comportement).
- Couvrir plusieurs APIs avec un seul harness (interleaved fuzzing).

Comment l'invoquer

- **Slash command** : `/harness-writing`
- **Phrases déclencheurs (texte)** : "write fuzz harness", "fuzz target", "LLVMFuzzerTestOneInput"
- **Auto-invocation** : Sur demande explicite

Description détaillée

Un harness de fuzzing, c'est le "point d'entrée" écrit par toi qui relie le fuzzer (qui génère des octets aléatoires) à ton vrai code (qui a une vraie API typée). En libFuzzer/AFL++, c'est typiquement une fonction `LLVMFuzzerTestOneInput(const uint8_t *data, size_t size)` : elle reçoit un paquet d'octets bruts et c'est à toi de les transformer en quelque chose que ta fonction comprend.

La qualité du harness détermine TOUT. Si tu fais juste `parser(data, size)` alors que ta vraie API attend une URL valide suivie d'un dict de headers, ton fuzzer va bombarder le parseur d'URL et jamais le code en aval. Inversement, un bon harness extrait intelligemment plusieurs valeurs typées des bytes (en utilisant `FuzzedDataProvider`), appelle plusieurs APIs dans un ordre cohérent, et fait du nettoyage entre chaque itération.

Les règles d'or : (1) reset l'état global entre chaque exécution (déterministe), (2) ne jamais quitter brutalement (`exit()` interdit), (3) libérer la mémoire allouée à chaque tour, (4) valider la taille d'entrée si nécessaire. Cette skill couvre les patterns pour C/C++, Rust (cargo-fuzz), Python (Atheris), Ruby (Ruzzy) et même les smart contracts. Tu peux faire un harness "single-API" (tester une fonction) ou "interleaved" (utiliser les premiers bytes pour choisir une opération parmi N).

Pour aller plus loin

Pour les exemples concrets, options de configuration et patterns avancés, voir le SKILL.md original.

Source

- **Plugin** : `trailofbits/testing-handbook-skills`
- **Nom interne** : `harness-writing`
- **Fichier** : `/home/thymon/.claude/plugins/cache/trailofbits/testing-handbook-skills/1.0.1/skills/harness-writing/SKILL.md`

Revision #2
Created 2026-05-11 21:19:58 UTC by thymon
Updated 2026-05-11 21:37:31 UTC by thymon