Eigenflow

GUIDE GRATUIT · SÉCURITÉ

Audite ton appli avec la skill de Cloudflare

Notre vrai test du 30/10 : 18 agents, 23 minutes, et ce qu'on en a fait

PAR

Abyane Nabeebaccus

INSTAGRAM

@abyane_nb

ÉDITION

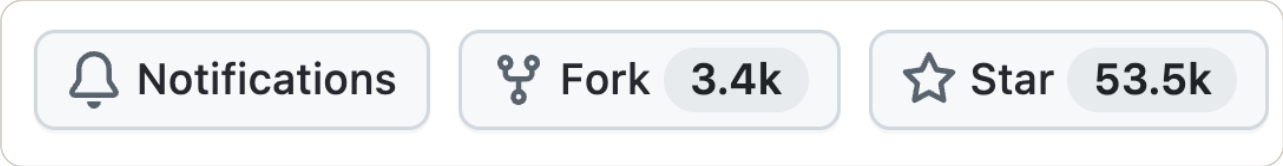
octobre 2026

Tu as commenté AUDIT. Voici ce qu'on a fait le 30 octobre 2026 entre 13 h 36 et 13 h 59 : installer la skill d'audit de sécurité de Cloudflare dans Claude Code, la lancer sur le code de notre propre hub (une appli Next.js de 77 fichiers, codée avec Claude), puis lire son rapport et agir. Toutes les commandes et tous les chiffres de ce guide viennent de cette session. Versions : Claude Code 2.1.280 avec Claude Opus 5.5, sur l'abonnement Claude (aucune clé d'API), skill au commit c1c8a8c (14 septembre 2026).

Ce qu'est cette skill

C'est un dossier de consignes que ton agent de code lit avant d'auditer ton projet. Cloudflare l'a publiée le 18 juin 2026, avec un billet de blog qui raconte comment ils l'ont utilisée sur leur propre code : l'outil qui en est né a sorti 7 245 problèmes à traiter dans leurs dépôts. Le dépôt est cloudflare/security-audit-skill, sous licence MIT : tu peux l'utiliser, la modifier, y compris pour un usage commercial, en gardant la mention de copyright.

Le 30 octobre, il avait 53 500 étoiles et 3 400 forks.



La page du dépôt le 30/10 : 53,5 k étoiles, 3,4 k forks.

Le déroulé tient en six phases, toujours dans le même ordre :

- **Reconnaissance** : des agents dressent la carte de ton appli (qui peut faire quoi, par où entrent les données, ce qui touche aux fichiers, à la base, aux autres services).

- **Chasse** : des agents « chasseurs », chacun sur un domaine (connexion, données, fichiers, IA…), cherchent à casser le code plutôt qu'à le relire.

- **Validation** : chaque piste passe à un agent neuf qui n'a pas chassé et qui essaie de prouver qu'elle n'existe pas.

- **Sortie structurée** : les pistes deviennent des fiches dans `findings.json`, vérifiées par deux petits programmes fournis avec la skill.

- **Revérification** : d'autres agents relisent les fiches finales.

- **Rapport** : `REPORT.md`, `FINDINGS-DETAIL.md` et `NEEDS-VALIDATION.md`, avec pour chaque problème la plus petite correction possible.

Ce qu'il te faut

- Claude Code, avec un abonnement Claude. La skill marche aussi avec Codex, Cursor, OpenCode et 75 autres agents, d'après son README ;

- Node.js 22.20 ou plus pour la commande d'installation : tape `node --version` pour vérifier ;

- le code de ton appli sur ton ordinateur.

Prévois du temps et du quota. Notre audit, en profil rapide, a lancé 18 agents en 23 minutes. Au prix de l'API, Claude Code l'a chiffré à 17,86 dollars ; avec l'abonnement, ça s'est imputé sur notre quota, sans facture en plus.

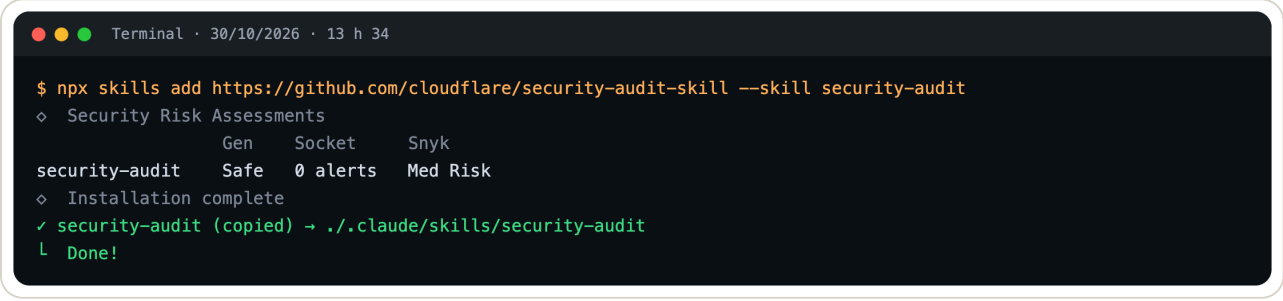
Étape 1 · Installer la skill

Dans le dossier de ton projet :

À COLLER

```
npx skills add https://github.com/cloudflare/security-audit-skill --skill
security-audit
```

Chez nous, l'installation a pris 4 secondes. L'installeur affiche son propre bilan avant de copier quoi que ce soit : « Safe » pour Gen, « 0 alerts » pour Socket, « Med Risk » pour Snyk. Il a copié la skill dans `.claude/skills/security-audit` et a fini sur un conseil qu'on te répète : relis une skill avant de t'en servir, elle tourne avec tous les droits de ton agent. On l'a fait : les 15 fichiers de consignes et les deux petits programmes de vérification (ils lisent des fichiers, rien d'autre, aucun accès réseau).



La sortie de l'installation chez nous : le bilan des trois analyseurs, puis la copie dans .claude/skills.

Étape 2 · La demande exacte

Ouvre Claude Code dans le dossier qui **contient** ton projet, pas dans le projet lui-même, et colle :

À COLLER

```
Fais un audit de sécurité complet du code dans mon-projet/ avec la skill
security-audit, profil quick, et écris le rapport dans audit/run-1.
```

Trois détails comptent, on les a appris en lisant la skill :

- **« audit de sécurité complet »** : sans ces mots, la skill reste en mode conseil. Elle répond à ta question, mais ne lance pas les six phases et n'écrit aucun rapport.

- **Le dossier du rapport doit être hors de ton projet.** Sinon, la skill s'arrête et te demande un autre chemin.

- **Le profil.** `quick` fait une seule vague de chasse : c'est un premier passage. `standard`, celui par défaut, suit tout le déroulé, avec plusieurs vagues et deux critiques de couverture. `deep`, pour les gros projets sensibles, relance des vagues jusqu'à ce qu'un critique ne trouve plus de trou.

Étape 3 · Ce qui se passe pendant l'audit

Voici notre session, minute par minute (heure de Paris) :

- 13:36 : Claude Code charge la skill et lit ses fichiers de méthode ;

- 13:37 : quatre agents de reconnaissance partent en même temps (le produit et sa pile, les droits, les points d'entrée, ce qu'on peut exécuter) ;

- 13:44 : cinq chasseurs partent, un par domaine : fichiers, connexion et CSRF, données et RLS, chat et rendu, divers (calendrier, secrets, fuites) ;

- 13:51 : les chasseurs ont rendu 8 pistes ;

- 13:52 : un critique de couverture relit la carte et note ce qui n'a pas été couvert ;

- 13:53 : huit vérificateurs, un par piste, essaient de démonter chacune ;

- 13:59 : rapport écrit, les deux programmes de vérification passent.

Tu n'as rien à faire pendant ce temps. Claude Code te dit où il en est, en français si tu lui as parlé en français.

```
Notre session · 18 agents · 23 min 20 s

13:36:07  Skill  cloudflare:security-audit
13:37:17  Agent  Recon 1a: product and stack
13:37:22  Agent  Recon 1b: principals and controls
13:37:28  Agent  Recon 1c: entry surfaces and sinks
13:37:36  Agent  Recon 1d: local execution visibility
13:44:14  Agent  Hunter h-files: file handling
13:44:18  Agent  Hunter h-access: auth and CSRF
13:44:22  Agent  Hunter h-data: RLS and business logic
13:44:26  Agent  Hunter h-ai: chat and rendering
13:44:29  Agent  Hunter h-misc: ICS, secrets, leakage
13:52:11  Agent  Final coverage critic (quick)
13:53:38  Agent  Verifier v-upload
13:53:42  Agent  Verifier v-noauth
13:53:45  Agent  Verifier v-csrf
13:53:48  Agent  Verifier v-chat-ctx
13:53:51  Agent  Verifier v-chat-img
13:53:54  Agent  Verifier v-ics
13:53:57  Agent  Verifier v-reel
13:54:00  Agent  Verifier v-telegram
13:59:24  REPORT.md, FINDINGS-DETAIL.md, NEEDS-VALIDATION.md écrits · validateurs OK
```

Les 18 agents de notre audit, avec l'heure de départ de chacun, relevés dans le journal de la session.

Étape 4 · Lire le rapport

Le rapport range chaque piste dans une de trois cases :

- **confirmée** : la faille est prouvée, avec sa gravité et sa correction ;

- **à valider** (needs validation) : le chemin dans le code est vérifié par un agent indépendant, mais un fait manque pour trancher. Pas de gravité ;

- **rejetée** : l'agent qui doutait a gagné.

Notre résultat : **aucune faille confirmée, 8 pistes à valider**. La skill refuse d'exécuter le code de ton appli sans un bac à sable qui coupe le réseau et limite la mémoire, le disque et le temps. Notre Mac ne fournissait pas tout ça : elle a donc tout vérifié en lisant le code, et a classé « à valider » ce qui demandait une exécution ou un fait absent du dépôt.

3. Findings confirmés

Sévérité	Titre	Frontière
—	*Aucun finding confirmé*	—

Aucune piste n'a pu être confirmée, faute de pouvoir l'exécuter de façon bornée dans ce run. Ce n'est pas une absence de risque.

5. NEEDS VALIDATION

8 pistes ont une trace source vérifiée par un agent indépendant. Elles n'ont **pas de sévérité** et ne sont pas des vulnérabilités confirmées. Le détail complet (trace, preuves, plans) est dans `NEEDS-VALIDATION.md`. Elles sont listées par ordre de priorité de levée de doute.

[tableau des 8 pistes : non reproduit ici]

Le début de notre rapport : la section des failles confirmées, vide, puis celle des pistes à valider. On ne reproduit pas le tableau des pistes : elles touchent notre propre serveur et ne sont pas toutes traitées.

Ne t'arrête pas à « aucune faille confirmée ». Le rapport le dit lui-même : « Ce n'est pas une absence de risque. » Pour chaque piste, il donne une colonne « Vérification par le propriétaire » :

ce que toi seul peux regarder.

Ce qu'on en a fait

Notre piste n° 1 disait en substance : aucune route de l'API ne demande d'identification, et tout dépend de qui peut joindre le serveur. Le dépôt ne le montre pas, nous si. On a vérifié à la main, en lecture seule : notre hub était en ligne, et son API répondait à n'importe qui, sans mot de passe.

On a corrigé le jour même :

- un mot de passe devant le hub, au niveau du serveur web (nginx), en laissant ouvert ce que nos outils doivent pouvoir lire ;

- la correction proposée par la skill pour la piste n° 2 : contrôler un identifiant reçu avant de s'en servir comme nom de dossier.

> **À COLLER**
>
> ```
> avant :  curl http://notre-serveur/api/leads  →  HTTP 200
> après :  curl http://notre-serveur/api/leads  →  HTTP 401
> ```

Les autres pistes sont dans notre liste, dans l'ordre de priorité du rapport.

Ce qu'il faut savoir

- **Un passage ne suffit pas.** Cloudflare l'écrit : un seul passage trouvait à peu près la moitié des failles trouvées en cumulant plusieurs passages. Relance la skill après tes corrections, elle relit les rapports précédents pour viser les trous.

- **Ne la lance jamais sur ce qui est en ligne.** Elle travaille sur le code. Ses consignes interdisent de sonder un site déployé, un service externe ou les données d'autres utilisateurs.

- **N'audite que ton code**, ou un code que tu as le droit d'auditer.

- **Ce n'est pas un pentest.** C'est une relecture outillée, très méthodique, qui te dit où regarder. Pour une appli qui manipule de l'argent ou des données sensibles, fais aussi appel à un professionnel.

Sources : dépôt cloudflare/security-audit-skill (README, SKILL.md, licence), billet « Build your own vulnerability harness » (blog Cloudflare, 18/06/2026), notre session du 30/10/2026.