Eigenflow

GUIDE · PLUGIN CLAUDE CODE

ECC dans Claude Code : les 71 agents, sans se faire piéger

Ce que le plugin installe vraiment sur ta machine, les deux commandes, les premiers ordres utiles, notre test de trois de ses agents avec Claude, et comment tout désinstaller.

PAR

Abyane Nabeebaccus

INSTAGRAM

@abyane_nb

ÉDITION

novembre 2026

Tu as commenté ECC, voici le guide. ECC est un plugin gratuit pour Claude Code, publié sous licence MIT par Affaan Mustafa sur github.com/affaan-m/ECC (l'ancienne adresse, everything-claude-code, redirige dessus). Il ajoute 71 agents, 314 skills, 98 commandes, des hooks, des règles, une mémoire entre les sessions et un scanner de configuration, AgentShield. Le dépôt comptait 299 919 étoiles le 9 novembre 2026 à 12 h 53.

D'abord ce qu'il contient et ce qu'il fait tourner sur ta machine. Ensuite les deux commandes d'installation, les premiers ordres qui servent vraiment, notre test, la désinstallation et les limites. Tout ce qui touche à ECC vient de son dépôt (version 2.2.3, dernier commit du 6 novembre 2026) et de la documentation officielle de Claude Code, relus le 9 novembre 2026. Eigenflow n'a aucun lien avec ECC ni avec son auteur.

À savoir avant de lire

- On n'a pas installé ECC. Ses hooks exécutent du code sur la machine, et on ne lance pas de code tiers pour une vidéo. Notre test donne à Claude le texte de trois de ses agents, sans plugin ni hook. Les parties sur l'installation viennent du README et de la doc de Claude Code.
- ECC est gratuit. Claude Code ne l'est pas : il faut un compte Pro, Max, Ultra, Team, Enterprise ou Console, le plan gratuit de claude.ai ne donne pas accès à Claude Code.
- Le plugin installe 27 commandes de hooks qui tournent avec tes droits d'utilisateur, en dehors du bac à sable de Claude Code.
- Installé en entier, ECC ajoute environ 48 079 tokens à chaque session, d'après l'estimation de Claude Code lui-même.
- Installe-le uniquement depuis les sources officielles. Le README prévient que des copies non officielles « may contain malware ».

---

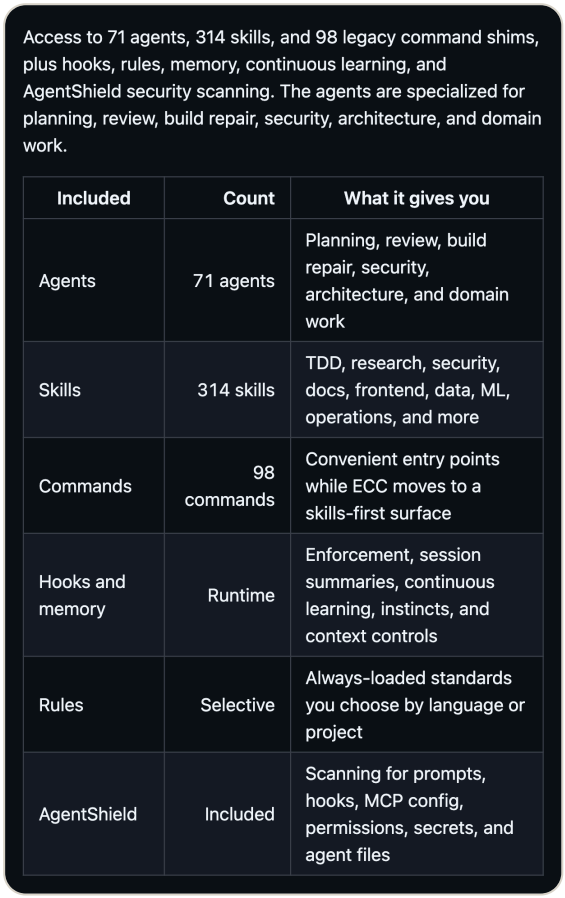
1. Ce qu'est ECC

Ce qu'il y a dedans

La phrase du README, au 9 novembre 2026 : « Access to 71 agents, 314 skills, and 98 legacy command shims, plus hooks, rules, memory, continuous learning, and AgentShield security scanning. » On a recompté dans le dépôt : 71 fichiers dans `agents/`, 314 dossiers dans `skills/`, 98 fichiers dans `commands/`.

- Les agents sont des sous-agents de Claude Code : une consigne, une liste d'outils, un modèle. 61 tournent sur Sonnet, 6 sur Haiku, 4 sur Opus (dont planner, qui fait les plans).

- Les skills sont des méthodes que Claude charge quand la tâche en a besoin : TDD, revue de sécurité, recherche, front, données. Beaucoup visent un langage ou un framework (Django, Laravel, Spring Boot, Kotlin, Rust…).
- Les commandes sont des raccourcis tapés avec une barre oblique. Le README les appelle « legacy » : le projet passe aux skills.
- Les hooks se déclenchent tout seuls, avant ou après une action de Claude. La partie 2 dit ce qu'ils font.
- Les règles sont des consignes chargées en permanence, par langage. Le plugin ne peut pas les livrer : elles se copient à la main.
- AgentShield audite la configuration de ton agent (CLAUDE.md, settings.json, serveurs MCP, hooks, agents, skills). Le code de ton appli reste hors de son champ.



Access to 71 agents, 314 skills, and 98 legacy command shims, plus hooks, rules, memory, continuous learning, and AgentShield security scanning. The agents are specialized for planning, review, build repair, security, architecture, and domain work.

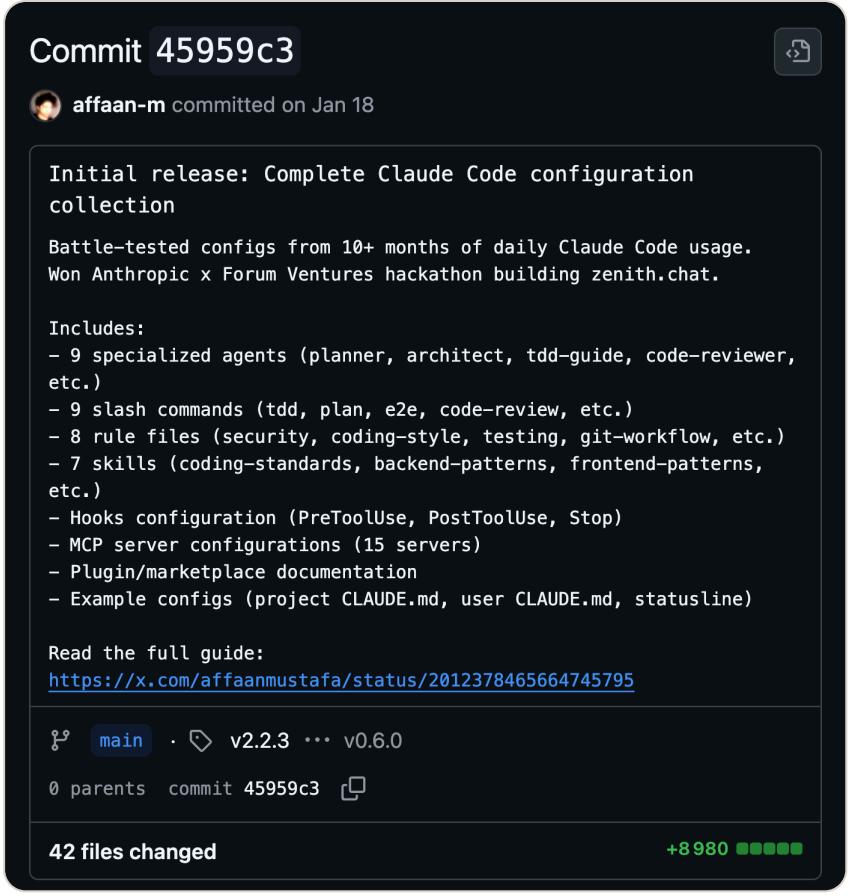
Included	Count	What it gives you
Agents	71 agents	Planning, review, build repair, security, architecture, and domain work
Skills	314 skills	TDD, research, security, docs, frontend, data, ML, operations, and more
Commands	98 commands	Convenient entry points while ECC moves to a skills-first surface
Hooks and memory	Runtime	Enforcement, session summaries, continuous learning, instincts, and context controls
Rules	Selective	Always-loaded standards you choose by language or project
AgentShield	Included	Scanning for prompts, hooks, MCP config, permissions, secrets, and agent files

Le README d'ECC le 9 novembre 2026 : 71 agents, 314 skills, 98 commandes, puis les hooks et la mémoire, les règles et AgentShield, avec ce que chaque partie apporte.

Qui l'a fait

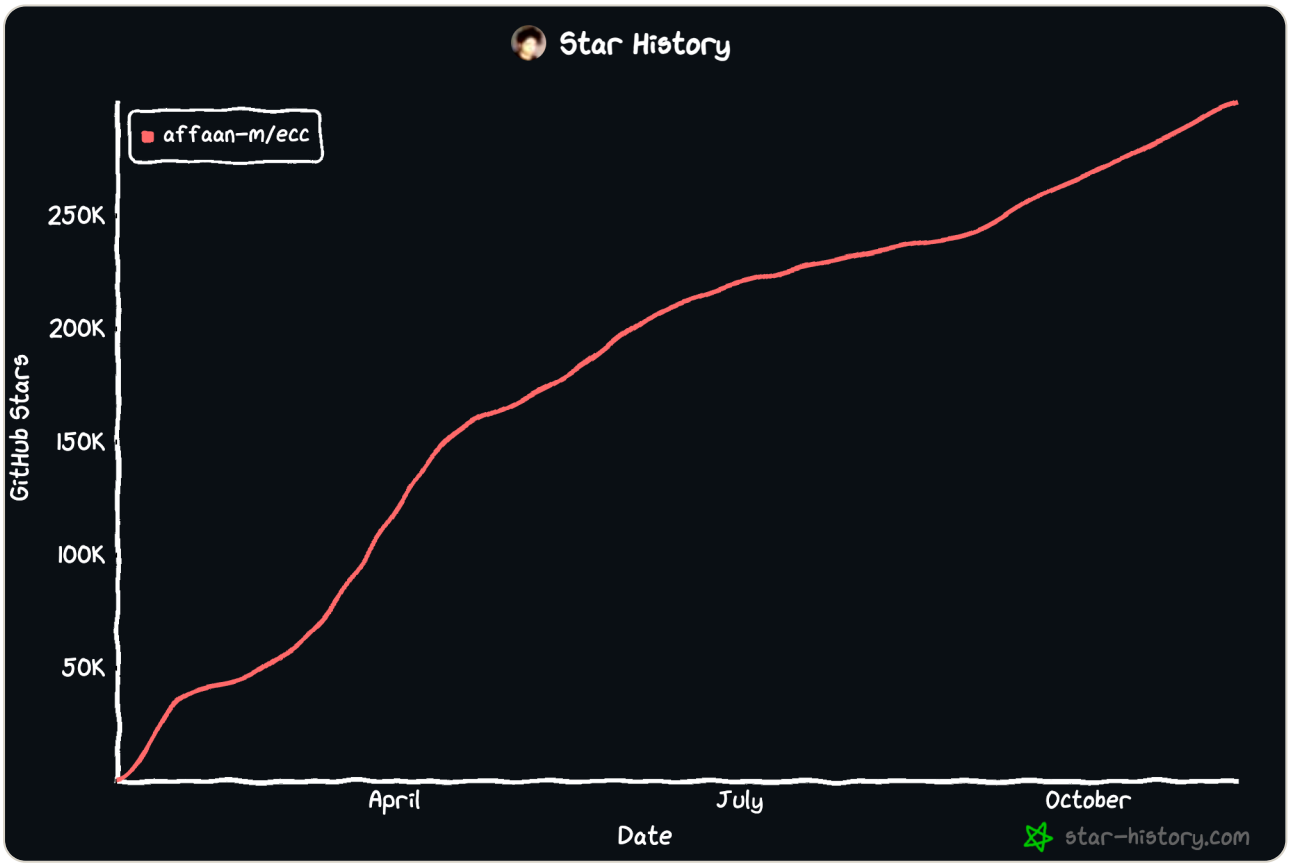
Affaan Mustafa a créé le dépôt le 18 janvier 2026. Son premier commit (45959c3, 42 fichiers) contenait 9 agents, 9 commandes, 8 fichiers de règles et 7 skills, avec ce message : « Battle-tested configs from 10+ months of daily Claude Code usage. Won Anthropic x Forum Ventures hackathon building zenith.chat. » Les dix mois d'usage quotidien, c'est sa déclaration ; on n'a pas de moyen de la vérifier.

Il a lancé le projet seul, et le README le dit encore : « a single maintainer ships weekly across 7 harnesses ». Depuis, d'autres ont contribué. Le 9 novembre, la page du dépôt affichait 486 contributeurs, et la branche principale comptait 4 327 commits, dont 1 778 signés par lui (41 %). Le dépôt avait 44 299 copies (forks).



Le premier commit d'ECC, le 18 janvier 2026 : 9 agents, 9 commandes, 8 règles, 7 skills, et la phrase sur les dix mois d'usage quotidien de Claude Code.

La courbe des étoiles monte de zéro en janvier à près de 300 000 en novembre, avec la pente la plus forte entre février et avril.



Étoiles GitHub d'ECC de janvier à novembre 2026, courbe de star-history.com relevée le 9 novembre 2026.

Ce qui est gratuit, ce qui se paie

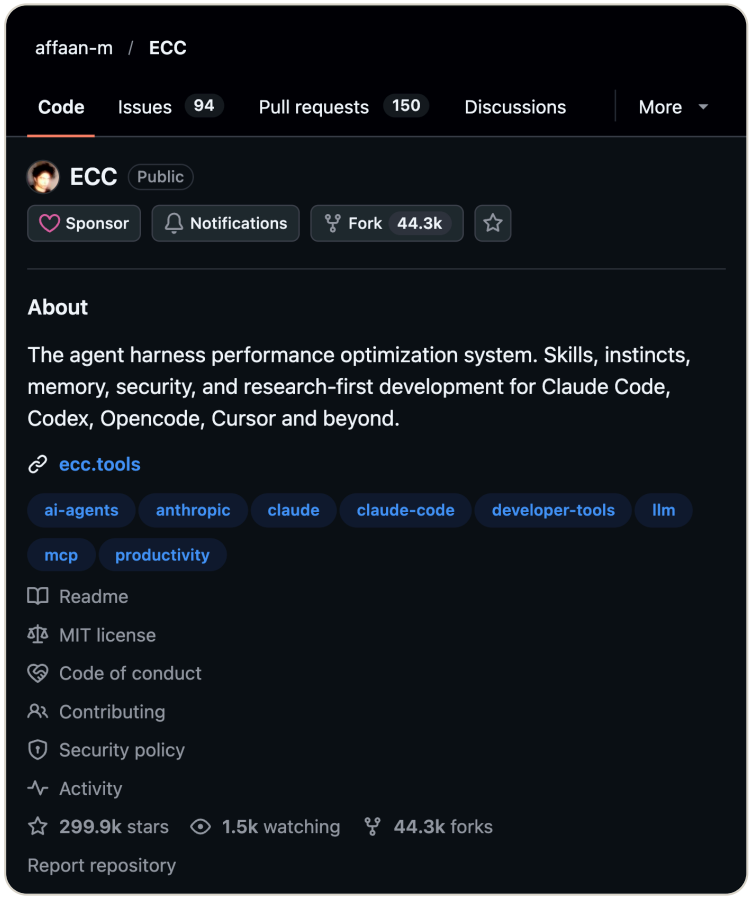
Le dépôt est sous licence MIT : tu peux l'utiliser, le modifier et le redistribuer, y compris pour un client, en gardant la mention de licence. Le README l'écrit ainsi : « This repo is MIT-licensed forever. »

À côté, il y a ECC Pro, une application GitHub hébergée qui analyse tes dépôts et lance des audits à chaque pull request. Sa page de prix (ecc.tools/pricing, 9 novembre 2026) donne trois formules :

- Free : 0 $, dépôts publics seulement, 10 analyses par mois.
- Pro : 19 $ par développeur actif et par mois, ou 190 $ par an, pour les dépôts privés, 50 analyses par siège.
- Enterprise : sur devis.

La page n'affiche aucun prix en euros et ne parle pas de TVA. Au cours de la BCE du 6 novembre 2026 (1 € = 1,1165 $), 19 $ font environ 17,02 € et 190 $ environ 170,18 €. Si une TVA française de 20 % s'ajoute au paiement, compte environ 20,42 € par mois ou 204,21 € par an. Ces conversions sont à nous ; le montant final se lit sur la page de paiement.

Ce qui coûte vraiment, c'est Claude Code. ECC ne fait que lui donner des consignes, et chaque agent consomme ta limite d'usage ou ta facture d'API.



La page GitHub d'ECC le 9 novembre 2026 : licence MIT, 299,9 k étoiles, 44,3 k forks, site ecc.tools.

2. Avant d'installer

Uniquement les sources officielles

Le README commence par un avertissement : « Third-party re-uploads and unofficial mirrors are not maintained or reviewed by the project and may contain malware. » Les sources officielles qu'il donne :

- le dépôt github.com/affaan-m/ECC ;
- le plugin `ecc@ecc` ;
- les paquets npm `ecc-universal` et `ecc-agentshield` ;
- l'application GitHub github.com/apps/ecc-tools ;
- le site ecc.tools.

Méfie-toi des noms qui ressemblent. Le README précise par exemple que `ecc-install` est un programme rangé dans le paquet `ecc-universal` : aucun paquet npm officiel ne porte ce nom. La documentation de sécurité d'ECC cite un cas concret : une campagne de mai 2026 où des paquets npm piégés plantaient des hooks dans `.claude/settings.json` pour voler des jetons GitHub.

Ce que les hooks lancent sur ta machine

La doc de Claude Code est claire : « A Claude Code plugin you install can execute arbitrary code on your machine with your user privileges. » Les hooks d'un plugin tournent hors du bac à sable, et tes règles de permission ne les arrêtent pas.

Chez ECC, `hooks/hooks.json` contient 27 commandes réparties sur 8 moments (avant et après chaque outil, à chaque message, au démarrage, à la fin de chaque réponse…). Toutes lancent `node`. Le profil par défaut, « standard », les active toutes. Voici ce qui change le plus ton quotidien, d'après les descriptions officielles des hooks :

- GateGuard refuse la première modification de chaque fichier tant que Claude n'a pas répondu à des questions sur ce qu'il touche. Les fichiers sensibles (secrets, clés, code d'authentification ou de paiement, migrations, CI) sont toujours concernés.
- Un autre hook bloque toute modification des fichiers de configuration du linter ou du formateur.
- Avant chaque commande Bash, un aiguilleur bloque `npm run dev` hors de tmux et vérifie tes `git commit` (lint, secrets, console.log).
- À la fin de chaque réponse, ECC reformate les fichiers JS et TS modifiés (Biome ou Prettier), lance `tsc`, cherche les console.log et t'envoie une notification macOS.
- Il garde une trace de ton travail sur le disque : résumés de session dans `~/.claude/session-data/`, coûts dans `~/.claude/metrics/costs.jsonl`, et une note par appel d'outil dans `~/.local/share/ecc-homunculus/` pour son « apprentissage continu ». L'observateur qui analyse ces notes en tâche de fond est désactivé par défaut.
- Au démarrage, il réinjecte le contexte de la session précédente, jusqu'à 8 000 caractères par défaut.

Tu peux régler tout ça. Le plugin propose deux réglages, « hooks_enabled » et « hook_profile » (minimal, standard ou strict), via `/plugin configure ecc@ecc`. Même en profil minimal, 12 des 27 commandes se lancent encore. Pour couper un hook précis, le README donne la variable `ECC_DISABLED_HOOKS`, par exemple `ECC_DISABLED_HOOKS="pre:bash:tmux-reminder"`.

Pour lire ce que le plugin contient avant de l'installer, la doc de Claude Code conseille d'ouvrir `hooks/hooks.json` et de lancer, sur une copie du dépôt, `claude --plugin-dir <dossier>`

`plugin details ecc`. Cette commande lit les fichiers sans ouvrir de session. C'est ce qu'on a fait.

Une seule installation par outil

Le README insiste : « Do not stack install methods. » Deux installations dans Claude Code dupliquent les skills, les commandes et les hooks, qui se déclenchent alors deux fois. Choisis un seul chemin :

- les deux commandes `/plugin` (partie 3) ;
- ou l'assistant `npx ecc-universal@2.2.3 setup`, qui installe le même plugin `ecc@ecc`.

Après le plugin, ne lance pas `./install.sh --profile full` et ne copie pas `hooks/hooks.json` dans ton `settings.json`.

Ce que ça coûte en contexte

Claude Code 2.1.280, sur une copie du dépôt, recense 412 skills (les 314 skills et les 98 commandes), 71 agents, 8 événements de hooks et un serveur MCP, et annonce « Always-on: ~48,079 tok added to every session ». Ce chiffre compte les noms et les descriptions de chaque skill, agent et commande, chargés à chaque session même si rien ne tourne. Dans la boutique officielle de plugins, Claude Code surligne ce coût dès 2 000 tokens.

Autre effet : Claude Code réserve à la liste des skills 1 % de la fenêtre de contexte du modèle. Au-delà, il retire des descriptions, en commençant par les skills que tu utilises le moins. Avec 412 entrées, beaucoup de skills d'ECC ne seront visibles que par leur nom, et Claude perd les mots-clés qui lui permettent de les choisir de lui-même. Le README le reconnaît : « use a selective/manual profile when context footprint matters ».

Après l'installation, regarde `/context` dans une session et `claude plugin details ecc` dans le terminal.

Par où commencer

Notre conseil : si ce sont les agents qui t'intéressent, copie les deux ou trois dont tu as besoin dans le dossier `.claude/agents/` d'un projet (la marche est en partie 5). Tu n'as alors aucun hook, et environ 250 tokens de contexte permanent pour planner, tdd-guide et security-reviewer réunis, d'après la même estimation de Claude Code. Installe le plugin entier si tu veux aussi ses hooks, sa mémoire et ses commandes, et fais-le d'abord dans un projet de test, en portée « local ».

---

3. Installer ECC dans Claude Code

Ce qu'il te faut

- Claude Code 2.1.0 ou plus récent (`claude --version`). Le README l'exige à cause de la façon dont les plugins chargent les hooks.
- Git.
- Node.js 18 ou plus récent, puisque chaque hook appelle `node`.

Les deux commandes

Ouvre Claude Code dans n'importe quel dossier et tape :

> /plugin marketplace add https://github.com/affaan-m/ECC

> /plugin install ecc@ecc

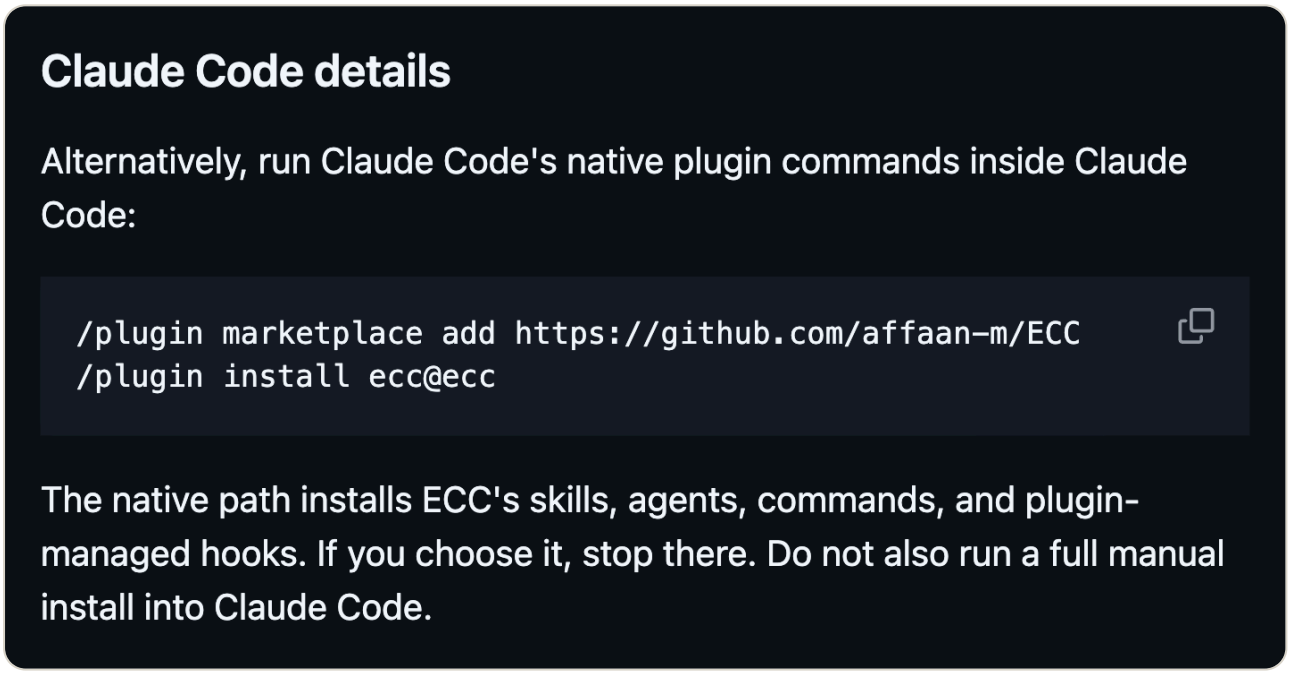
La première ajoute le catalogue d'ECC à Claude Code. La seconde ouvre d'abord la fiche du plugin, avec la liste de ce qu'il va installer et un avertissement de confiance. Tu choisis ensuite la portée :

1. pour toi, dans tous tes projets (user) ;

2. pour tous les collaborateurs de ce dépôt (project), qui l'écrit dans `.claude/settings.json` ;

3. pour toi, dans ce dépôt seulement (local).

Pour un premier essai, prends « local » dans un projet de test. Le résumé se termine par « Plugin is now active. » ou par « Run /reload-plugins to apply. ».



La section « Claude Code details » du README d'ECC : les deux commandes, et la consigne de s'arrêter là sans ajouter d'installation manuelle.

Vérifier

> /plugin list ecc@ecc

La ligne doit montrer la version (2.2.3 au 9 novembre 2026), la portée et l'état. Dans le terminal, `claude plugin list` donne la même chose, et `claude plugin details ecc` affiche l'inventaire avec le coût en tokens. Tape `/ecc:` dans une session : la liste des skills et commandes d'ECC apparaît.

Les règles, à part

Les plugins de Claude Code ne peuvent pas livrer de règles. Si tu les veux, le README conseille de copier le dossier commun et un seul langage, celui que tu utilises :

À COLLER

```
git clone https://github.com/affaan-m/ECC.git
mkdir -p ~/.claude/rules/ecc
cp -R ECC/rules/common ~/.claude/rules/ecc/
cp -R ECC/rules/typescript ~/.claude/rules/ecc/
```

Remplace `typescript` par ton langage (21 dossiers disponibles, de `python` à `golang` en passant par `vue`). Ces règles sont chargées à chaque message : le dossier commun fait à lui seul 10 fichiers et 19 408 caractères.

L'autre chemin : l'assistant npx

> npx ecc-universal@2.2.3 setup

Le README le présente comme le chemin recommandé. Il installe le même plugin `ecc@ecc`, te fait choisir la portée et le profil de hooks, et sert aussi aux mises à jour. Pour un premier usage personnel, le README conseille « Global user » et des hooks « Standard ». Il faut Node.js 18 ou plus. Le README ajoute lui-même : « A version pin is not a security audit or an integrity check. » Cette commande télécharge et exécute le paquet npm.

Codex et Cursor

- Codex a un plugin natif : `codex plugin marketplace add affaan-m/ECC` puis `codex plugin add ecc@ecc`. Ses hooks demandent une validation explicite dans `/hooks`, les profils de hooks de Claude ne s'appliquent pas, et les fichiers d'agents ne deviennent pas des rôles Codex.
- Cursor passe par un adaptateur en bêta, installé depuis une copie du dépôt avec `./install.sh --profile minimal --target cursor`. Il écrit dans le dossier `.cursor/` du projet, et le README prévient que la découverte des agents « varies by Cursor build ».

---

4. Les premiers ordres utiles

Le préfixe ecc:

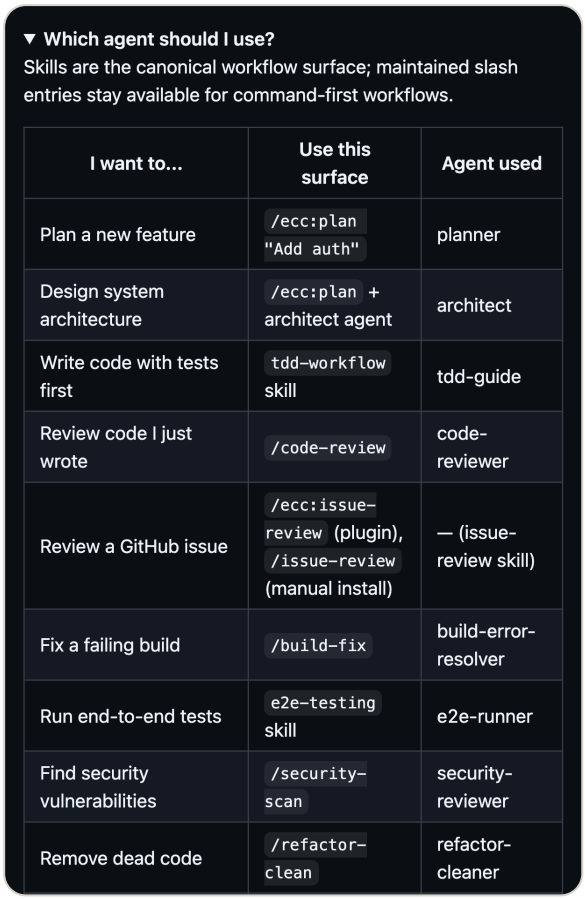
Dans Claude Code, les skills et commandes d'un plugin se tapent avec le nom du plugin devant : `/ecc:plan`, `/ecc:code-review`. Le README écrit souvent la forme courte (`/code-review`), valable pour une installation manuelle. Avec le plugin, garde le préfixe : Claude Code a déjà ses propres `/plan` (le mode plan), `/code-review` et `/security-review`, qui ne sont pas ceux d'ECC.

Ce que le README conseille

Le tableau « Which agent should I use? » associe chaque besoin à une commande et à l'agent qui travaille derrière. Les plus utiles pour commencer :

- Planifier une fonctionnalité : `/ecc:plan`, agent planner.
- Écrire le code, tests d'abord : skill `tdd-workflow`, agent tdd-guide.

- Relire ce que tu viens d'écrire : `/ecc:code-review`, agent code-reviewer.
- Réparer un build cassé : `/ecc:build-fix`, agent build-error-resolver.
- Tests de bout en bout : skill `e2e-testing`, agent e2e-runner.
- Supprimer le code mort : `/ecc:refactor-clean`, agent refactor-cleaner.



▼ **Which agent should I use?**
Skills are the canonical workflow surface; maintained slash entries stay available for command-first workflows.

I want to...	Use this surface	Agent used
Plan a new feature	`/ecc:plan "Add auth"`	planner
Design system architecture	`/ecc:plan` + architect agent	architect
Write code with tests first	`tdd-workflow` skill	tdd-guide
Review code I just wrote	`/code-review`	code-reviewer
Review a GitHub issue	`/ecc:issue-review` (plugin), `/issue-review` (manual install)	— (issue-review skill)
Fix a failing build	`/build-fix`	build-error-resolver
Run end-to-end tests	`e2e-testing` skill	e2e-runner
Find security vulnerabilities	`/security-scan`	security-reviewer
Remove dead code	`/refactor-clean`	refactor-cleaner

Le tableau « Which agent should I use? » du README d'ECC, relevé le 9 novembre 2026 (début du tableau).

Une fonctionnalité, de bout en bout

La suite « Starting a new feature » du README, adaptée à une vraie demande :

> /ecc:plan "Ajoute la connexion par e-mail et mot de passe, avec sessions de 24 h"

`/ecc:plan` reformule la demande, liste les risques, découpe en étapes, puis attend ton accord avant de toucher au code. Il planifie dans la conversation, sans sous-agent : sa consigne dit « Do not call the Task tool or any subagent by default ». Si tu veux l'agent planner sur Opus, demande-le :

> Délègue ce plan à l'agent planner d'ECC, puis montre-moi le plan avant d'écrire du code.

Une fois le plan validé :

> /ecc:tdd-workflow

Puis la relecture :

> /ecc:code-review

Avant de mettre en production

La suite « Preparing for production » enchaîne `/security-scan`, `e2e-testing` et `/test-coverage`. Un piège dans le premier : le tableau du README associe `/security-scan` à « Find security vulnerabilities », mais le fichier de la commande lance AgentShield, qui audite ta

configuration d'agent (hooks, MCP, permissions, secrets). Pour faire relire le code de ton appli par l'agent de sécurité, demande-le par son nom :

> Utilise l'agent security-reviewer d'ECC sur auth.js : liste les failles avec leur gravité et la ligne, puis propose le code corrigé.

Et pour la couverture de tests :

> /ecc:test-coverage

Elle détecte ton outil de test (Jest, Vitest, pytest, cargo, Maven, go test), liste les fichiers sous 80 % de couverture, écrit les tests manquants et recommence tant que le seuil n'est pas atteint.

Pour les longues sessions

- `/ecc:context-budget` renvoie vers le compteur de contexte de Claude Code et, si tu le demandes, fait l'inventaire de ce qu'ECC y charge.
- `/ecc:save-session` enregistre où tu en es, `/ecc:resume-session` reprend plus tard.

---

5. Notre test, sans installer ECC

La méthode

Le 9 novembre 2026, on a pris le texte de trois agents d'ECC (planner, tdd-guide, security-reviewer) et de la commande test-coverage, en-tête retiré. Chaque texte a servi de consigne système à Claude Code en mode non interactif : `claude -p --safe-mode --tools ""`, sur notre abonnement, dans un dossier vide, sans plugin, sans hook, sans outil. Claude ne pouvait que répondre ; un script enregistrait ses réponses et lançait les tests.

La demande : la connexion par e-mail et mot de passe d'une appli de prise de rendez-vous pour un salon de coiffure. Contraintes : Node.js 22, aucune dépendance (seulement `node:crypto`, `node:test`, `node:assert`), un module `auth.js`, un stockage en mémoire, pas de serveur HTTP.

1. Planifie : consigne de planner, modèle Opus (claude-opus-5-5).

2. Construis : consigne de tdd-guide, modèle Sonnet (claude-sonnet-5).

3. Sécurise : consigne de security-reviewer, Sonnet.

4. Teste : consignes de test-coverage et de tdd-guide, Sonnet, en boucle jusqu'à 80 % des lignes et des branches sans test en échec, quatre tours au plus.

Le plan

Opus a rendu un plan de 38 lignes : 6 étapes, 3 fichiers (`auth.js`, `auth.test.js`, `package.json`), 5 risques avec leur parade. Il a pensé à vérifier l'unicité de l'adresse une seconde fois après le hachage, et à hacher contre un sel factice quand l'adresse est inconnue pour ne pas trahir les comptes existants par le temps de réponse. Un extrait, mot pour mot :

À COLLER

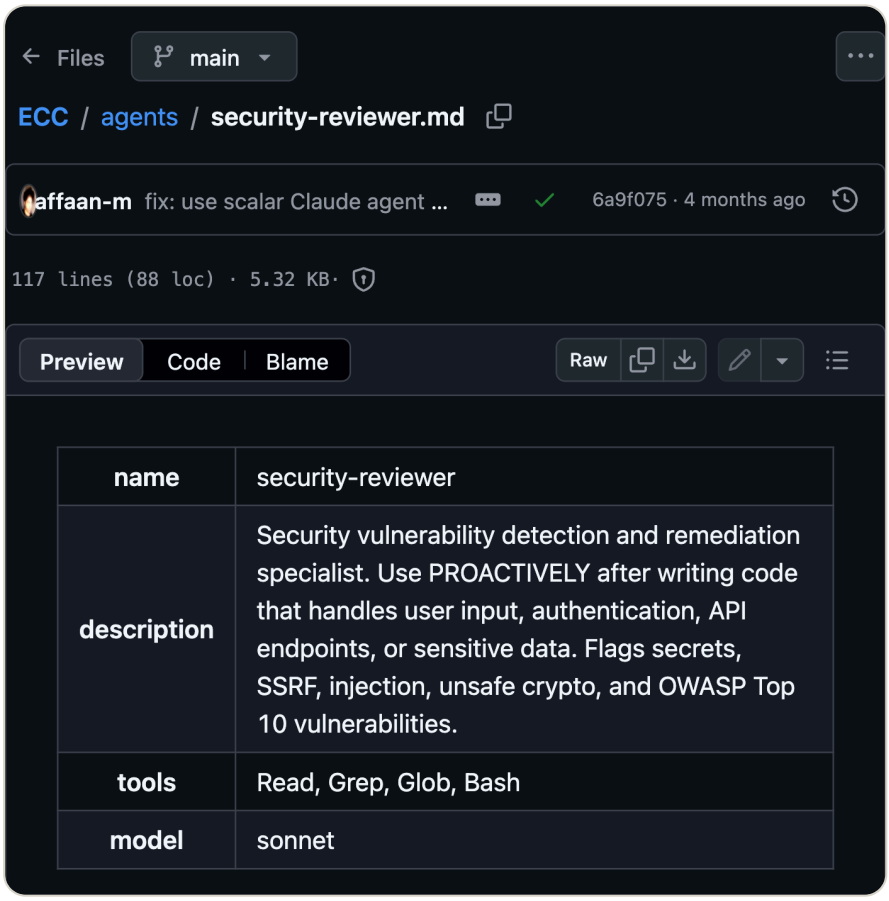
```
**Risques et parades**
- **Énumération de comptes** par le message ou le temps de réponse → erreur générique et hachage factice (étape 4).
- **Force brute** : rien ne limite les essais → compteur d'échecs par e-mail dans une `Map`, en phase 2 si besoin.
- **Croissance mémoire** : les sessions expirées s'accumulent → purge paresseuse dans `login` et `verifySession`.
```

Ce plan a pris 869 secondes et représente 3,47 $ des 3,96 $ du test, au tarif de l'API.

Le code

Avec la consigne de tdd-guide, Sonnet a écrit `auth.js` (127 lignes) et `auth.test.js` (19 tests). Les 19 passent du premier coup, rien n'a été corrigé à la main. Il a repris le plan (hachage factice, seconde vérification de l'adresse), sans le compteur d'échecs que le plan repoussait « en phase 2 ».

La revue de sécurité



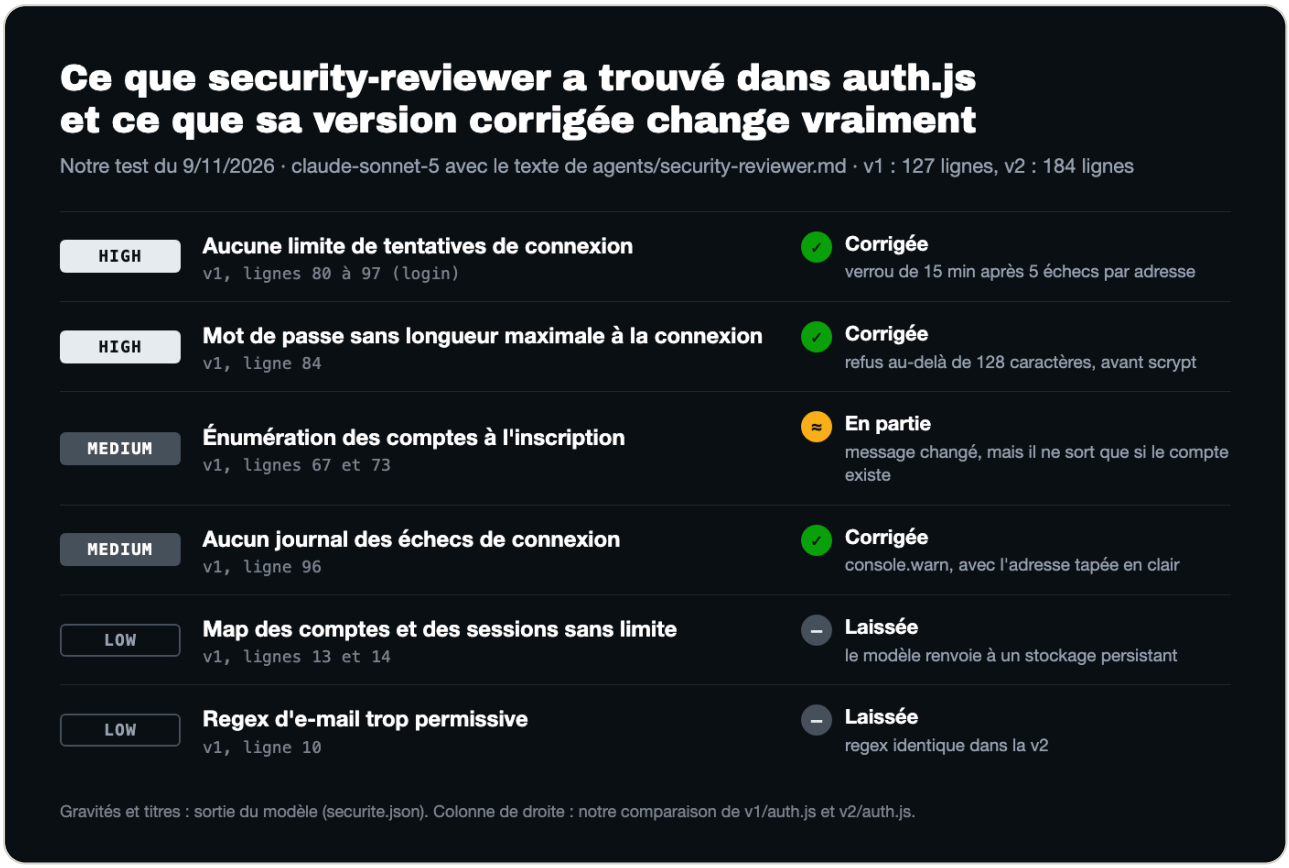
La fiche de l'agent security-reviewer sur GitHub, le 9 novembre 2026 : « detection and remediation specialist », à lancer de lui-même après du code qui touche aux entrées, à l'authentification ou aux données sensibles. Ses outils sont Read, Grep, Glob et Bash, sans droit d'écriture, et il tourne sur Sonnet.

Avec la consigne de security-reviewer, Sonnet a listé 6 failles : 2 HIGH, 2 MEDIUM, 2 LOW, aucune CRITICAL. La première, telle qu'il l'a écrite :

À COLLER

```
{"gravite": "HIGH", "ligne": "80–97 (login)",
 "titre": "Absence de limitation des tentatives de connexion",
 "explication": "Rien n'empêche un attaquant de tester un nombre illimité de
mots de passe par force brute ou credential stuffing contre login()."}
```

Il a aussi rendu une version corrigée, `v2/auth.js`, de 184 lignes. On l'a comparée ligne à ligne à la première. Sur les 6 failles, 3 sont vraiment corrigées, 1 l'est en apparence, 2 restent.



Les 6 failles trouvées dans auth.js et ce que la version corrigée change, d'après notre comparaison des deux fichiers.

Pour l'énumération, la v2 change seulement le message :

À COLLER

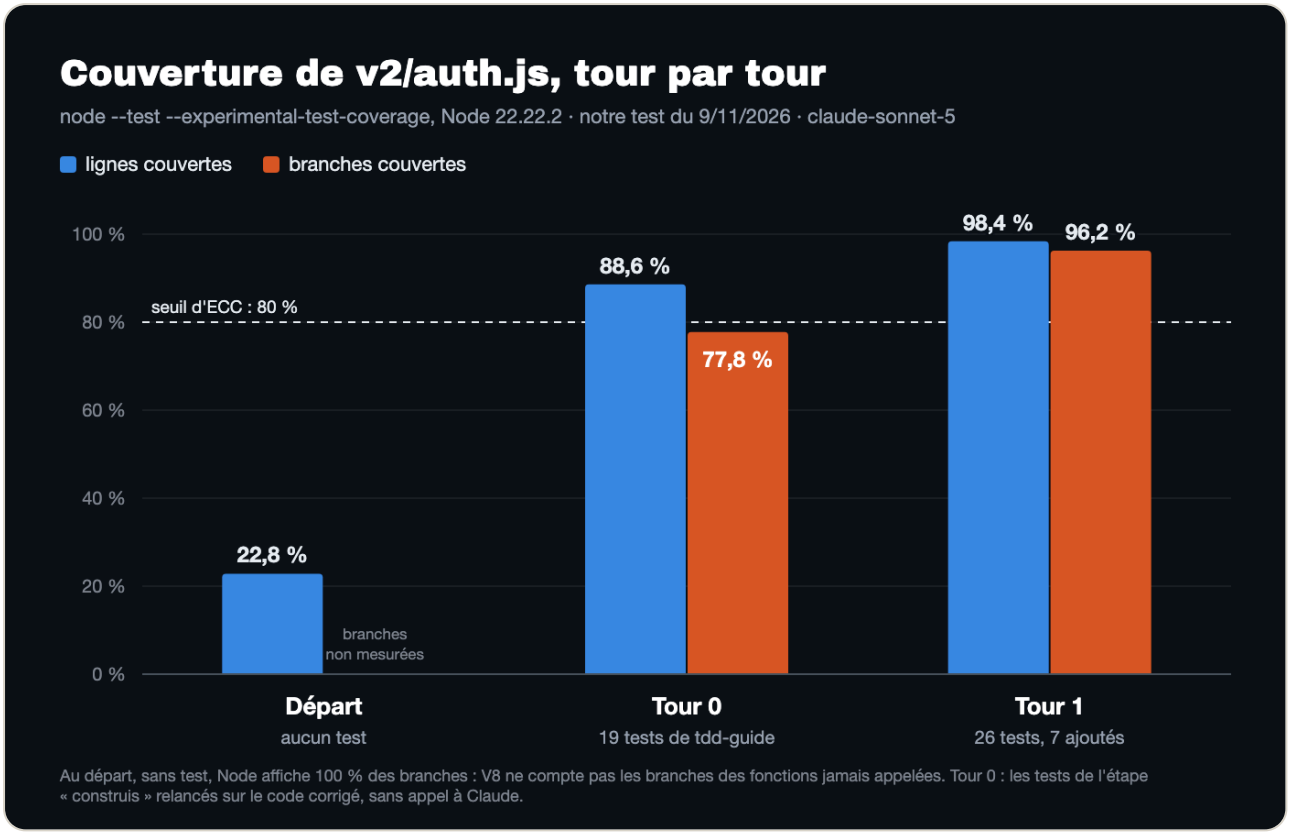
```
v1 :  throw new Error('Cette adresse e-mail est déjà utilisée.');
v2 :  throw new Error(REGISTRATION_FAILED_MESSAGE);
      // "Impossible de créer le compte avec ces informations."
```

Le nouveau message ne sort que si le compte existe. Avec une adresse et un mot de passe valides, l'inscription réussit ou renvoie ce message, donc un attaquant sait toujours quelles adresses sont inscrites. Le verrou contre la force brute a aussi un effet de bord, que le modèle signale lui-même : cinq mauvais essais suffisent pour bloquer le compte de quelqu'un d'autre pendant 15 minutes. Et le journal des échecs écrit l'adresse tapée en clair dans la console.

La couverture

On a lancé `node --test --experimental-test-coverage` (Node 22.22.2) sur le code corrigé. Avec les 19 tests de l'étape 2, on était à 88,59 % des lignes et 77,78 % des branches, sous le

seuil. Un seul tour de Claude a suffi : 7 tests ajoutés (verrouillage après 5 échecs, purge des tentatives au bout d'une heure, mot de passe trop long, deux inscriptions simultanées…), 26 tests, 26 passent.



Couverture de v2/auth.js au départ, avec les tests de l'étape 2, puis après le tour de Claude.

La sortie brute de Node, fin du rapport :

```
# tests 26
# pass 26
# fail 0
# file         | line % | branch % | funcs % | uncovered lines
# auth.js      |  98.37 |    96.23 |  100.00 | 169-171
```

Ce qu'on en retient

Le test a coûté 4 appels à Claude, 1 237 secondes au total et 3,96 $ au tarif de l'API, payés par l'abonnement.

Les consignes d'ECC donnent des réponses carrées. Le plan nomme ses risques, et la revue classe les failles par gravité avec la ligne en cause. Elle a repéré la force brute que le plan avait repoussée à plus tard. Elle a aussi rendu une correction incomplète, et rien ne l'a signalé : franchement, c'est ce qui nous a le plus surpris. Les 26 tests passent, la couverture dépasse 98 %, et l'énumération des comptes reste possible. Une couverture de 98 % dit seulement que les tests ont fait tourner 98 % des lignes.

Ce test laisse de côté tout ce que le plugin ajoute autour des agents : hooks, skills, mémoire, lancement en sous-agents. Et il tient sur une seule fonctionnalité, avec un seul lancement par étape.

Refaire le test chez toi, sans installer ECC

Il te faut Claude Code et Node.js 22. Ces commandes téléchargent seulement le texte des agents, elles n'exécutent rien qui vienne d'ECC :

À COLLER

```
mkdir ecc-test && cd ecc-test
base=https://raw.githubusercontent.com/affaan-m/ECC/main
curl -sfLO $base/agents/planner.md
curl -sfLO $base/agents/tdd-guide.md
curl -sfLO $base/agents/security-reviewer.md
curl -sfLO $base/commands/test-coverage.md
for f in planner tdd-guide security-reviewer test-coverage; do sed '1,/^---$/d'
$f.md > $f.txt; done
cat test-coverage.txt tdd-guide.txt > couverture.txt
```

Étape 1, le plan, avec nos mots exacts :

À COLLER

```
echo "Planifie cette fonctionnalité : la connexion par e-mail et mot de passe
pour l'appli de prise de rendez-vous d'un salon de coiffure. Contraintes :
Node.js 22, sans framework ni aucune dépendance (seulement node:crypto,
node:test, node:assert) ; un module auth.js qui exporte des fonctions
(inscription, connexion, vérification de session) ; stockage en mémoire (un Map)
; pas de serveur HTTP. Réponds en français, plan court (étapes, fichiers,
risques), 40 lignes au plus." | claude -p --safe-mode --model opus --tools "" --
no-session-persistence --append-system-prompt-file planner.txt > plan.md
```

Étape 2, le code. Copie ensuite les deux blocs de la réponse dans `auth.js` et `auth.test.js`, puis lance `node --test` :

À COLLER

```
{ echo "Voici le plan :"; echo; cat plan.md; echo; echo "Construis-la : écris
d'abord les tests puis le code. Réponds uniquement avec deux blocs de code,
auth.js puis auth.test.js. Contraintes : Node.js 22, sans framework ni aucune
dépendance (seulement node:crypto, node:test, node:assert) ; un module auth.js
qui exporte des fonctions (inscription, connexion, vérification de session) ;
stockage en mémoire (un Map) ; pas de serveur HTTP."; } | claude -p --safe-mode
--model sonnet --tools "" --no-session-persistence --append-system-prompt-file
tdd-guide.txt > construis.md
```

Étape 3, la revue de sécurité. Notre script demandait la réponse en JSON pour la lire automatiquement ; ici, la version lisible :

À COLLER

```
{ echo "Voici le fichier auth.js :"; echo; cat auth.js; echo; echo "Sécurise-la : liste les failles (gravité CRITICAL/HIGH/MEDIUM/LOW, ligne, en une phrase en français), puis le fichier corrigé complet."; } | claude -p --safe-mode --model sonnet --tools "" --no-session-persistence --append-system-prompt-file security-reviewer.txt > securite.md
```

Avant de remplacer `auth.js` par la version corrigée, compare les deux fichiers toi-même (`diff`), faille par faille.

Étape 4, la couverture. Relance ce bloc, en remplaçant chaque fois `auth.test.js` par la réponse, jusqu'à 80 % des lignes et des branches sans échec :

À COLLER

```
node --test --experimental-test-coverage auth.test.js > rapport.txt
{ echo "Voici auth.js :"; cat auth.js; echo "Voici les tests actuels, auth.test.js :"; cat auth.test.js; echo "Rapport de couverture brut :"; cat rapport.txt; echo "Teste-la : ajoute les tests manquants pour dépasser 80 % de couverture. Réponds uniquement avec le fichier auth.test.js complet."; } | claude -p --safe-mode --model sonnet --tools "" --no-session-persistence --append-system-prompt-file couverture.txt > tests.md
```

Ces appels passent par ton abonnement et entament ta limite d'usage. Les options sont celles de Claude Code 2.1.280 ; tape `claude --help` si l'une manque chez toi.

Plus simple : les trois agents comme sous-agents, sans le plugin

Claude Code charge les fichiers d'agents posés dans `.claude/agents/` d'un projet. Tu obtiens les vrais sous-agents, avec leurs outils et leur modèle, sans hooks ni skills :

À COLLER

```
mkdir -p ecc-agents/.claude/agents && cd ecc-agents
base=https://raw.githubusercontent.com/affaan-m/ECC/main/agents
curl -sfL $base/planner.md -o .claude/agents/planner.md
curl -sfL $base/tdd-guide.md -o .claude/agents/tdd-guide.md
curl -sfL $base/security-reviewer.md -o .claude/agents/security-reviewer.md
claude
```

Puis, dans la session :

> Utilise l'agent planner pour planifier la connexion par e-mail et mot de passe de mon appli de rendez-vous : Node.js 22, aucune dépendance, un module auth.js, stockage en mémoire.

> Utilise l'agent tdd-guide pour construire auth.js et auth.test.js selon ce plan, tests d'abord.

> Utilise l'agent security-reviewer sur auth.js : liste les failles avec leur gravité et la ligne, puis propose le code corrigé.

> Lance node --test --experimental-test-coverage et ajoute des tests jusqu'à 80 % des lignes et des branches, sans aucun test en échec.

tdd-guide et security-reviewer ont accès à Bash : Claude Code te demandera la permission avant chaque commande.

Avec ECC installé

> /ecc:plan "Connexion par e-mail et mot de passe pour une appli de rendez-vous, Node.js 22, sans dépendance"

> /ecc:tdd-workflow

> Utilise l'agent security-reviewer d'ECC sur auth.js, puis montre-moi le diff de la correction.

> /ecc:test-coverage

Attends-toi à ce que GateGuard refuse la première écriture de `auth.js` et pose ses questions : le code d'authentification fait partie des fichiers toujours contrôlés.

---

6. Désinstaller proprement

Dans une session :

> /plugin uninstall ecc@ecc

Ou dans le terminal, avec la portée choisie à l'installation :

> claude plugin uninstall ecc@ecc --scope local

Pour retirer aussi le catalogue d'ECC (ce qui désinstalle tout plugin venu de lui) :

> /plugin marketplace remove ecc

Ce qui reste après, d'après la doc de Claude Code et le code des hooks :

- Les fichiers du plugin restent 14 jours dans `~/.claude/plugins/cache/` avant un nettoyage automatique. Pour les effacer tout de suite, supprime `~/.claude/plugins/cache/ecc/ecc/`.
- Les traces écrites par les hooks vivent hors du dossier du plugin : `~/.claude/session-data/`, `~/.claude/metrics/costs.jsonl`, `~/.local/share/ecc-homunculus/`. Regarde ce qu'ils contiennent avant de les supprimer.
- Les règles copiées à la main : `rm -rf ~/.claude/rules/ecc`.

Si tu es passé par l'assistant npx, désinstalle depuis le dossier où tu l'as lancé, en regardant d'abord ce qui sera retiré :

À COLLER

```
npx ecc-universal@2.2.3 uninstall --dry-run
npx ecc-universal@2.2.3 uninstall
```

Si tu as empilé plusieurs installations, le README donne cet ordre : retirer le plugin de Claude Code, lancer la désinstallation d'ECC depuis le dossier qui contient son état d'installation, supprimer les règles copiées à la main, puis réinstaller une seule fois par un seul chemin.

---

7. Les limites, dites franchement

On ne l'a pas fait tourner

Ce guide décrit les hooks d'après leur code et leur documentation, sans les avoir vus tourner. Notre test porte sur trois consignes d'agents.

Le README ne colle pas toujours aux fichiers

Deux exemples relevés le 9 novembre 2026. Le tableau du README dit que `/security-scan` sert à trouver les failles et passe par security-reviewer ; le fichier de la commande lance AgentShield sur la configuration de l'agent. Le même tableau dit que `/ecc:plan` passe par l'agent planner ; le fichier de la commande planifie dans la conversation et n'appelle planner que si tu le demandes. Avec 4 327 commits en dix mois, ce genre d'écart arrive. Lis le fichier d'une commande avant de compter dessus.

71 agents, 71 consignes

L'« équipe » tient dans 71 fichiers texte. Leur qualité dépend du modèle qui les lit et de ce que tu leur demandes. Dans notre test, la revue de sécurité a vu juste sur la liste et s'est arrêtée à mi-chemin sur une correction.

Le contexte

Environ 48 000 tokens par session, plus les règles que tu ajoutes, plus jusqu'à 8 000 caractères réinjectés au démarrage. Sur un abonnement, ça veut dire des limites atteintes plus vite. Le README lui-même propose une installation minimale quand le contexte compte.

Les mises à jour

Pour un catalogue tiers comme celui d'ECC, la mise à jour automatique est désactivée par défaut dans Claude Code. Tu restes sur ta version tant que tu ne lances pas `claude plugin update ecc@ecc`. Si tu l'actives, les fichiers que tu as relus peuvent changer sans que tu le voies : relis `hooks/hooks.json` à chaque grosse version.

Un seul mainteneur

Le README le dit : un seul mainteneur publie chaque semaine pour 7 outils. Le projet dépend beaucoup d'une personne, financée par des sponsors et par ECC Pro.

Les systèmes

Le README signale des défauts ouverts sous Windows natif (l'observateur de l'apprentissage continu, l'écriture de la mémoire) et sur le Bash 3.2 de macOS pour une fonction optionnelle.

Sources

- Dépôt ECC, README, fichiers `.claude-plugin/plugin.json`, `hooks/hooks.json`, `hooks/hooks.metadata.json`, `hooks/README.md`, `agents/planner.md`, `agents/tdd-guide.md`, `agents/security-reviewer.md`, `commands/plan.md`, `commands/test-coverage.md`, `commands/security-scan.md`, commit d2ef268 du 6 novembre 2026, relus le 9 novembre 2026 : github.com/affaan-m/ECC
- Premier commit, 18 janvier 2026 : github.com/affaan-m/ECC/commit/45959c3
- API GitHub du dépôt, 9 novembre 2026 à 11 h 53 UTC : 299 919 étoiles, 44 299 forks, licence MIT.
- Prix d'ECC Pro, 9 novembre 2026 : ecc.tools/pricing
- Cours de référence de la BCE du 6 novembre 2026 : ecb.europa.eu
- Claude Code, découvrir et installer des plugins : code.claude.com/docs/en/discover-plugins
- Claude Code, sécurité des plugins : code.claude.com/docs/en/plugins/security
- Claude Code, commandes des plugins et `plugin details` : code.claude.com/docs/en/plugins/cli-reference
- Claude Code, coût d'un plugin : code.claude.com/docs/en/plugins/measure
- Claude Code, skills et budget de la liste : code.claude.com/docs/en/skills
- Claude Code, comptes requis : code.claude.com/docs/en/setup