



Run an AI security review before you ship.

A step-by-step way to have AI adversarially pen-test your own app before it goes live — and hand you a clear PASS or BLOCK. This is the exact gate every build of mine clears first, generalized so you can run it on yours.

by Corina • The CV Method • @the_cv_method • current
as of Aug 2026 — verify tool details, this space
moves fast

RULE ZERO

00

**Assume it's broken until proven safe.
BLOCK by default.**

Nothing deploys until the gate passes — or until you consciously waive a specific item, in writing, with a reason. Safe-by-default is the whole point; "probably fine" is not a verdict.



A repeatable pre-deploy gate: you point an AI at your own app, tell it to attack — hunt for the ways in the way a real intruder would — fix what it finds, and end with a written **PASS/BLOCK** verdict before anything launches.

It isn't a scanner you buy or a certificate you frame. It's a disciplined pass any small team can run, on tools you already have. It rests on one rule, so we start there.

STEP 1 • PICK YOUR STACK

01

You don't need a security vendor to start.

Four simple pieces — here's what I use and the general equivalent.

PIECE 01

The reviewer

An AI that can read your code and reason like an attacker.

I USE **Claude**

SWAP Any strong tool-use LLM

PIECE 02

The runner

Where the AI actually reads your repo and works through it.

I USE **Claude Code / an agent runner**

SWAP Paste files into a capable chat

PIECE 03

The target

Your app code + its config — the thing going live.

I USE **A repo (e.g. a Worker/app folder)**

SWAP Whatever your app lives in

PIECE 04

The output

A written review with evidence and a verdict.

I USE **A SECURITY_REVIEW.md in the repo**

SWAP Any doc you keep with the code



Tell it to assume it's broken

A HELPER MISSES WHAT AN ATTACKER LOOKS FOR

Frame the AI as a strict, adversarial penetration tester and tell it to **assume at least three critical mistakes exist** until proven otherwise. For each check it should state PASS / FAIL / N/A with file-and-line evidence; for every FAIL, exactly how it'd be exploited and the precise fix. Keep the framing prompt saved so every review starts identically — consistency is what makes it a gate, not a vibe.

The mindset is the whole trick. Ask an AI "is this secure?" and it reassures you. Tell it to break in, and it starts finding the doors you left open.

The seven things to hunt for

- | | | |
|---|-----------------------------------|---|
| 1 | Access & authorization | every permission decision made server-side, never trusting the browser; users can't reach another user's data by changing an ID (no IDOR); sessions time-limited and revocable. |
| 2 | Secrets & data | no keys or credentials in the code or anywhere in git history; secrets read from the environment at runtime; personal data only where a screen truly needs it. |
| 3 | App hardening | input validated on both client and server; standard security headers (CSP, HSTS, no-sniff, no-store); no SQL-injection or XSS surface; CORS not left wide open. |
| 4 | Rate limits & abuse | limits on login, password-reset, any intake, and any AI endpoint; failed-auth logging. |
| 5 | Signed intake | (if you receive data from another system) — the trigger is cryptographically signed and verified before anything is processed, and re-sends are idempotent. |
| 6 | AI endpoints | treat all user and retrieved content as untrusted (prompt-injection resistant); fail to a safe fallback, never a hard crash. |
| 7 | Health & monitoring | a public health check and real error capture from day one. |

CRITICAL blocks **HIGH** blocks unless waived with a logged reason

MEDIUM tracked follow-up

Tag every finding. The tag decides what happens next.



PASS or BLOCK, with the evidence

A VERDICT YOU CAN POINT TO LATER

BLOCK

if any Critical item fails, or any High item fails without a logged waiver.

PASS

otherwise, with Medium items recorded as follow-ups. No "mostly done."

Write it into a short review doc kept beside the code: a one-paragraph summary, a findings table (item · priority · status · evidence · fix), the open items, any conscious waivers with who approved them, and the Medium follow-ups. That doc is what "we did security" actually looks like — and it's your honest answer when a client asks "how secure is this?"

Legal/compliance items (AI disclosure, privacy, any HIPAA/SOC-2 claim) get flagged for a human — never self-certified by the AI or by you.

Clear it, run it again, then deploy

A PASS IS A MOMENT, NOT A MEDAL

Fix each finding, then re-run the whole gate — fixes create new gaps, and a partial re-check is how the quiet ones slip through. Only deploy when it comes back clean, and deploy behind a step that itself can't leak (an allowlist build — a different playbook).

Keep the review doc in the repo. It's your audit trail, your onboarding note for a teammate, and the thing that turns "trust me" into "here's the pass." Re-run the gate on every meaningful change, not just the first launch. Security is a habit, not a launch task.

KEEP IT HONEST & SAFE

- You own the fix & the ship decision
- No self-certifying — legal items go to a human
- Fixes verified, not assumed
- Teach the method, never a client's keys/config
- Least access for the reviewer
- No invented "we're compliant" claims



The adversarial review — paste, pointed at your app

```
Act as a strict, adversarial penetration tester auditing [my app/repo] before launch.  
Assume at least three critical mistakes exist until proven otherwise.  
Work the gate: access & authorization (server-side only, no IDOR, scoped data); secrets (none in  
code  
or git history; env only); hardening (input validated both sides; CSP/HSTS/no-store; no SQLi/XSS;  
CORS  
locked); rate limits (login/reset/intake/AI endpoints); signed intake (HMAC-verified, idempotent);  
AI endpoints (untrusted input; safe fallback); health & monitoring.  
For each item: PASS / FAIL / N/A with file+line evidence; for every FAIL, how it'd be exploited and  
the  
exact fix. Then write SECURITY_REVIEW.md and give me a one-line verdict — PASS (no Critical/High  
open)  
or BLOCK. Do not deploy.
```

THE STARTER CHECKLIST

10 BOXES

- | | |
|--|--|
| ☐ Frame the AI as an adversarial pen-tester; "assume 3 critical bugs." | ☐ External intake is signed (HMAC) and idempotent. |
| ☐ Access: authorization server-side only, no IDOR, scoped data. | ☐ AI-endpoint input treated as untrusted; safe fallback. |
| ☐ Secrets: none in code or git history; env only. | ☐ Public health check + error capture on. |
| ☐ Hardening: input validated both sides; CSP/HSTS/no-store; no SQLi/XSS; CORS locked. | ☐ Write the verdict: PASS (no Critical/High open) or BLOCK. |
| ☐ Rate limits on login, reset, intake, and AI endpoints. | ☐ Fix, re-run the whole gate, then deploy behind a leak-proof step. |



The CV Method

Want your build audited — or the whole gate set up for your team?

This is the adversarial pre-deploy review every system I ship clears first. I'll run it on yours, or stand up the gate so your team runs it on every release.

hello.thecvmethod.com →