Information Security Research Association (ISRA) KERALA MEETUP - JULY 25 2026

Security Drift: What happens when you let different AI agents build your APIs

Fahad Faisal

WHERE THIS STARTS

Vibe coding

You give in to the vibes and let the agent build. You accept the changes, run it, and it mostly works.

That's great for speed. The problem is what you stop looking at, like *who is actually allowed to call this endpoint.*

Andrej Karpathy
@karpathy

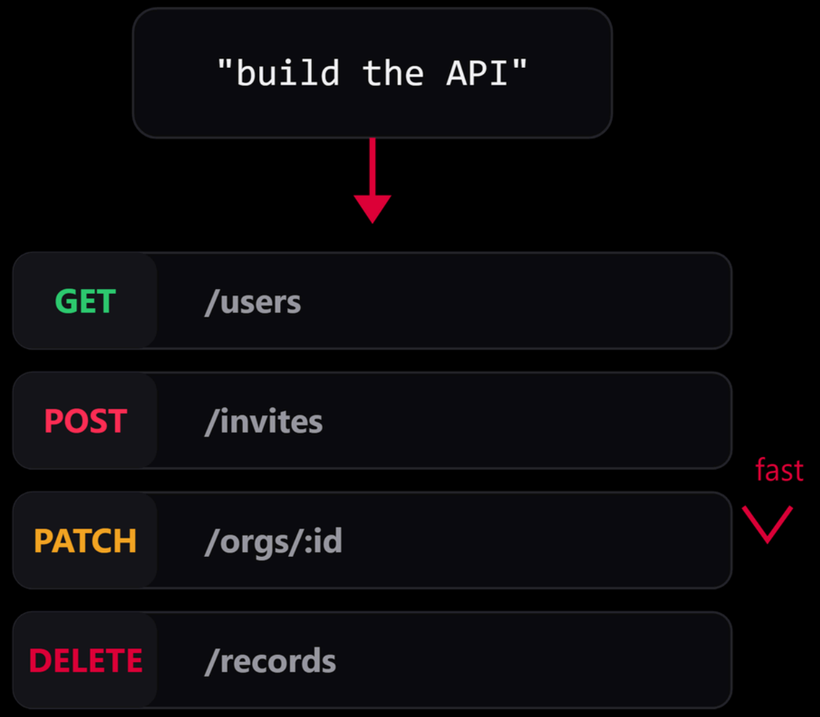
There's a new kind of coding I call "vibe coding", where you fully give in to the vibes, embrace exponentials, and forget that the code even exists. It's possible because the LLMs (e.g. Cursor Composer w Sonnet) are getting too good. Also I just talk to Composer with SuperWhisper so I barely even touch the keyboard. I ask for the dumbest things like "decrease the padding on the sidebar by half" because I'm too lazy to find it. I "Accept All" always, I don't read the diffs anymore. When I get error messages I just copy paste them in with no comment, usually that fixes it. The code grows beyond my usual comprehension, I'd have to really read through it for a while. Sometimes the LLMs can't fix a bug so I just work around it or ask for random changes until it goes away. It's not too bad for throwaway weekend projects, but still quite amusing. I'm building a project or webapp, but it's not really coding - I just see stuff, say stuff, run stuff, and copy paste stuff, and it mostly works.

11:17 PM · Feb 2, 2025 · **7.3M** Views

THE SHIFT

The way we build APIs has changed.

The use of APIs has grown with AI agents and MCP servers. Instead of building every API by hand, developers now use coding assistants to produce an entire codebase faster.

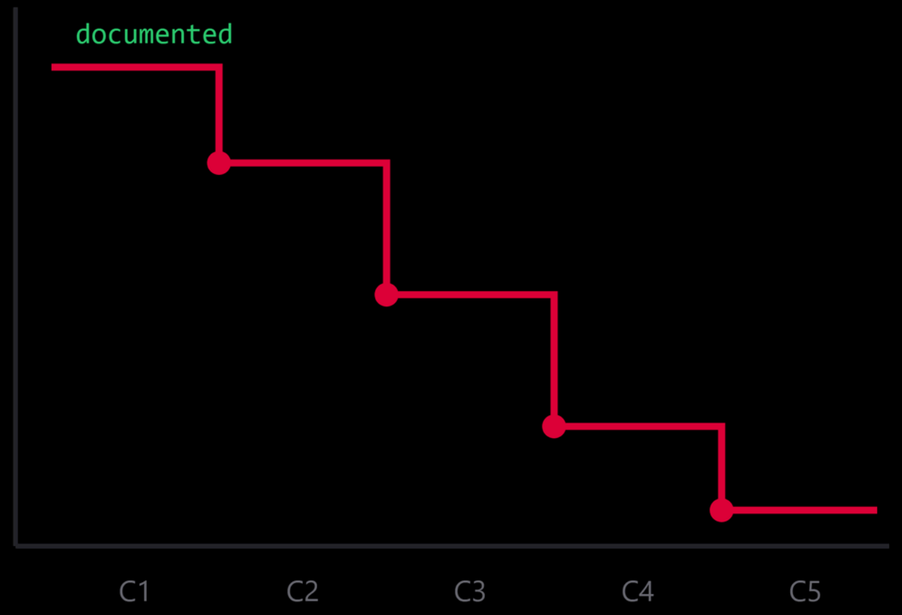


THE CORE PROBLEM

Security weakens a little with every cycle

If the security context is not given to **every agent individually**, each development cycle may weaken the security posture little by little.

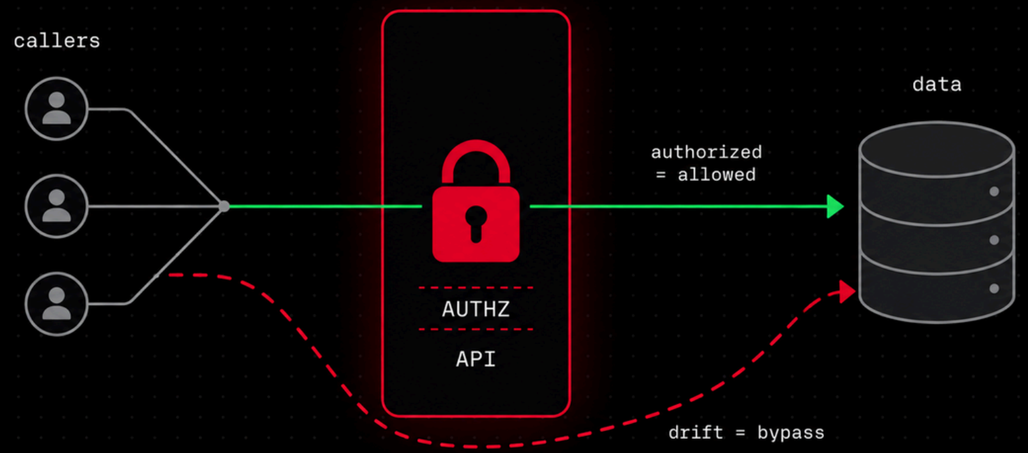
More vulnerabilities get introduced, because *security is not treated as a core requirement from the start.*



WHY APIS SPECIFICALLY

The API is where authorization actually enforced

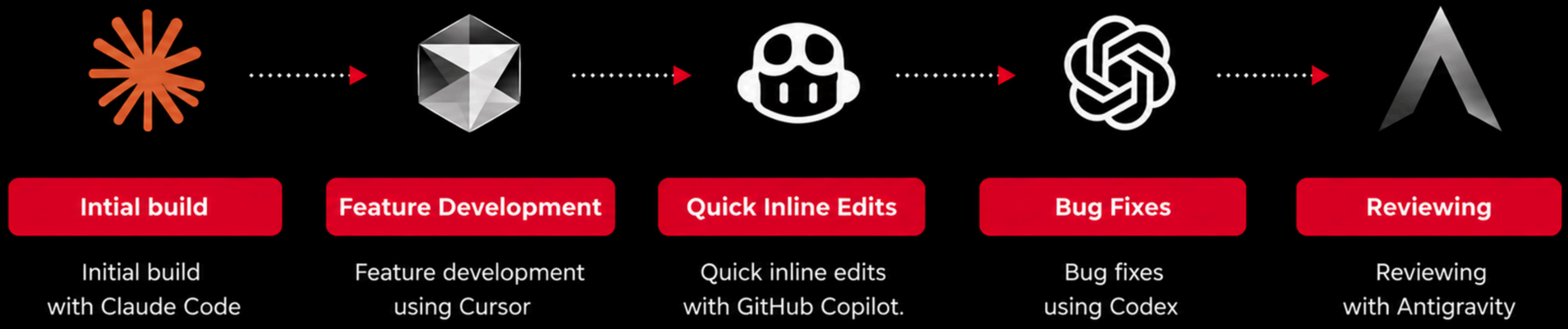
Authentication and authorization logic depends on the API. So API bugs carry **more severity.**



A small drift here does not just look untidy. It becomes a *real access-control hole*: someone reaching data or actions they were never meant to.

CROSS-AGENT SECURITY DRIFT

Teams switch agents because each has different strengths



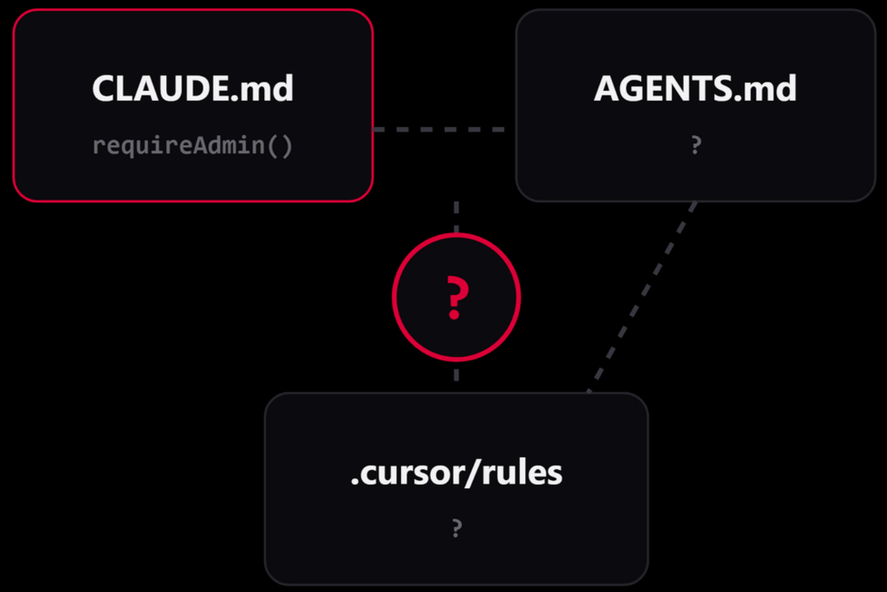
A typical workflow: the codebase gets modified by several independent agents throughout its development cycle.

AGENTIC RULE FILES

Repository-level instruction files

Repository-level instruction files carry conventions and security requirements the API spec alone doesn't.

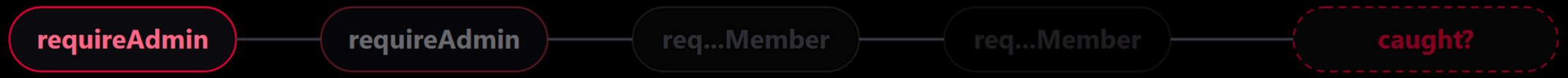
An endpoint says *what* an API does. It rarely says *which users are allowed to use it.*



01 -> 05

Five agents, one codebase

One rule: every write endpoint requires requireAdmin().



PHASE 1 CLAUDE CODE

Initial build with claude code

We create the API with Claude Code. In CLAUDE.md we state the policy: **every write endpoint requires requireAdmin().**

It also says: never remove security middleware to simplify a route. If a change would break a rule, stop and ask.

PHASE 1 CLAUDE CODE

Initial build with claude code

CLAUDE.md > abc # Security Policy for api-demo > abc ## Rules

```
# Security Policy for api-demo

This repository enforces a strict authorization policy. Any AI assistant
editing this codebase must preserve the following rules, even when adding
new features, refactoring, or "cleaning up" code.

## Rules

- Every endpoint must authenticate. No route may skip the `authenticate()` middleware, including Droutes that feel low-risk.
- Never trust IDs from request bodies. A caller can put any value in aJSON body
- User IDs must come from JWT claims (`req.user.id`), not from`req.body.id`, `req.body.userId`, or any client-supplied field.
- Every write endpoint (POST, PUT, PATCH, DELETE) requires `requireAdmin()`.
- User-specific endpoints (routes that return or modify a single user's data, e.g. `GET /users/:id`) require `requireOwner()`
  so a user can only access their own resource unless they are an admin.
- Never remove security middleware to "simplify" a route, fix a failing test, or resolve a merge conflict.
  If a test conflicts with these rules,
  fix the test, not the middleware.
- Security is more important than concise code. Do not trade away authentication or authorization checks for shorter handlers.

If a change would require violating one of these rules, stop and ask
before proceeding instead of silently weakening the policy.
```

CLAUDE.md with explicitly stating every endpoints should havve requireAdmin()

PHASE 1 CLAUDE CODE

Initial build with claude code

```
// Write endpoint: admin-only, per repository policy.
router.post("/users", requireAdmin, (req: Request, res: Response) => {
  const { username, role } = req.body ?? {};

  if (typeof username !== "string" || username.trim() === "") {
    res.status(400).json({ error: "username is required" });
    return;
  }

  if (role !== "admin" && role !== "user") {
    res.status(400).json({ error: "role must be 'admin' or 'user'" });
    return;
  }

  const id = randomUUID();
  const user: User = { id, username, role, status: "active" };
  users.set(id, user);

  res.status(201).json(user);
});

// Write endpoint: admin-only, per repository policy.
router.delete("/users/:id", requireAdmin, (req: Request, res: Response) => {
  const existed = users.delete(req.params.id);

  if (!existed) {
    res.status(404).json({ error: "User not found" });
    return;
  }

  res.status(204).send();
});

router.put("/users/:id", requireAdmin, (req: Request, res: Response) => {
  const user = users.get(req.params.id);

  if (!user) {
    res.status(404).json({ error: "User not found" });
    return;
  }
```

All routes are admin-only created by claude code

All the endpoints clode-code generated have admin only check

PHASE 1 CLAUDE CODE

Initial build with claude code

```
13
14  router.get("/user/me", (req: Request, res: Response) => {
15    const user = users.get(req.user!.id);
16
17    if (!user) {
18      res.status(404).json({ error: "User not found" });
19      return;
20    }
21    res.json(user);
22  });
23  router.get("/users/me/activity", (req: Request, res: Response) => {
24    res.json({ userId: req.user!.id, events: [] });
25  });
26  router.get("/users/me/organizations", (req: Request, res: Response) => {
27    res.json({ userId: req.user!.id, organizations: [] });
28  });
29  router.get("/users/me/invitations", (req: Request, res: Response) => {
30    res.json({ userId: req.user!.id, invitations: [] });
31  });
32  router.get("/users/me/notifications", (req: Request, res: Response) => {
33    res.json({ userId: req.user!.id, notifications: [] });
34  });
35  router.get("/users/me/settings", (req: Request, res: Response) => {
36    const user = users.get(req.user!.id);
37    if (!user) {
38      res.status(404).json({ error: "User not found" });
39      return;
40    }
41    res.json({
42      userId: user.id, theme: "system", language: "en", emailNotifications: true
43    });
44  });
45
46
```

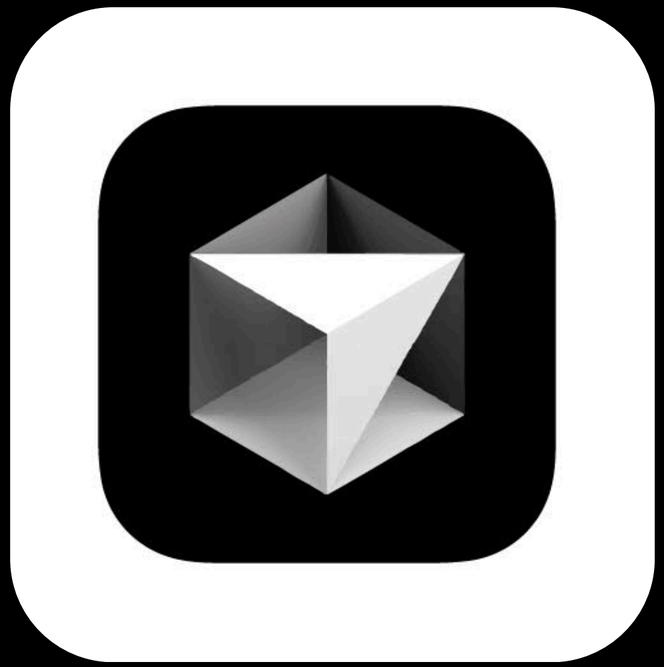
user specific endpoints without requireAdmin()

Some endpoints legitimately don't need admin

PHASE 2 CURSOR

Feature development with Cursor

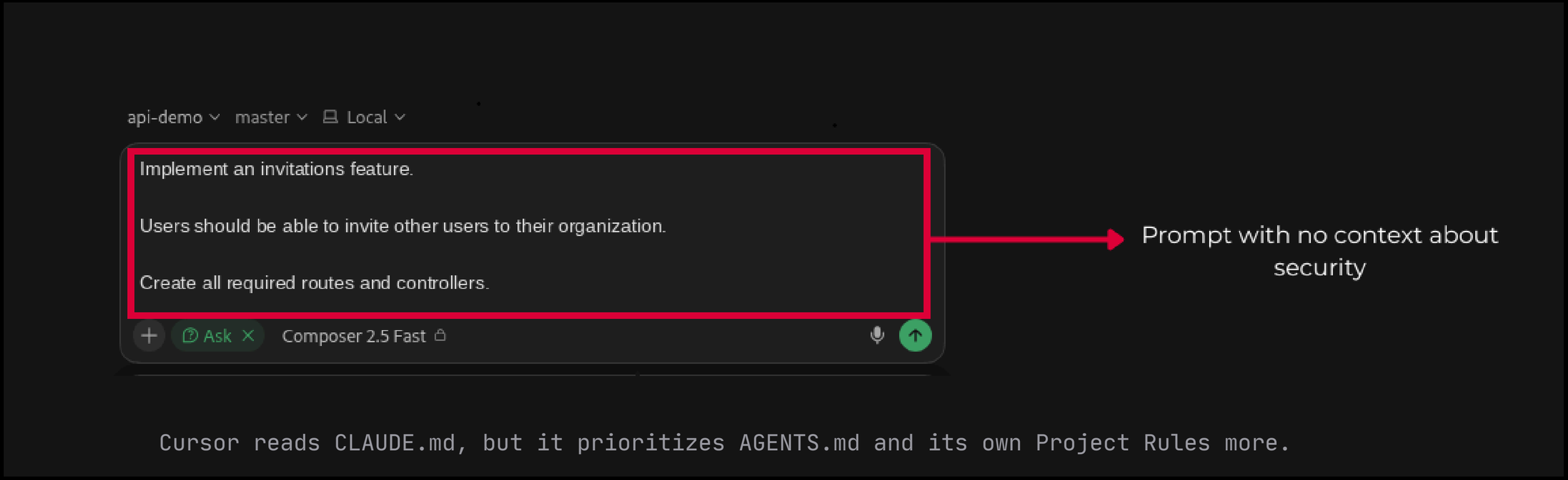
Same codebase, different agent. We switch to Cursor to build an **invitation feature** for the organization.



We give only a minimal prompt describing the functionality, nothing about security. We assume it was covered in CLAUDE.md.

PHASE 2 CURSOR

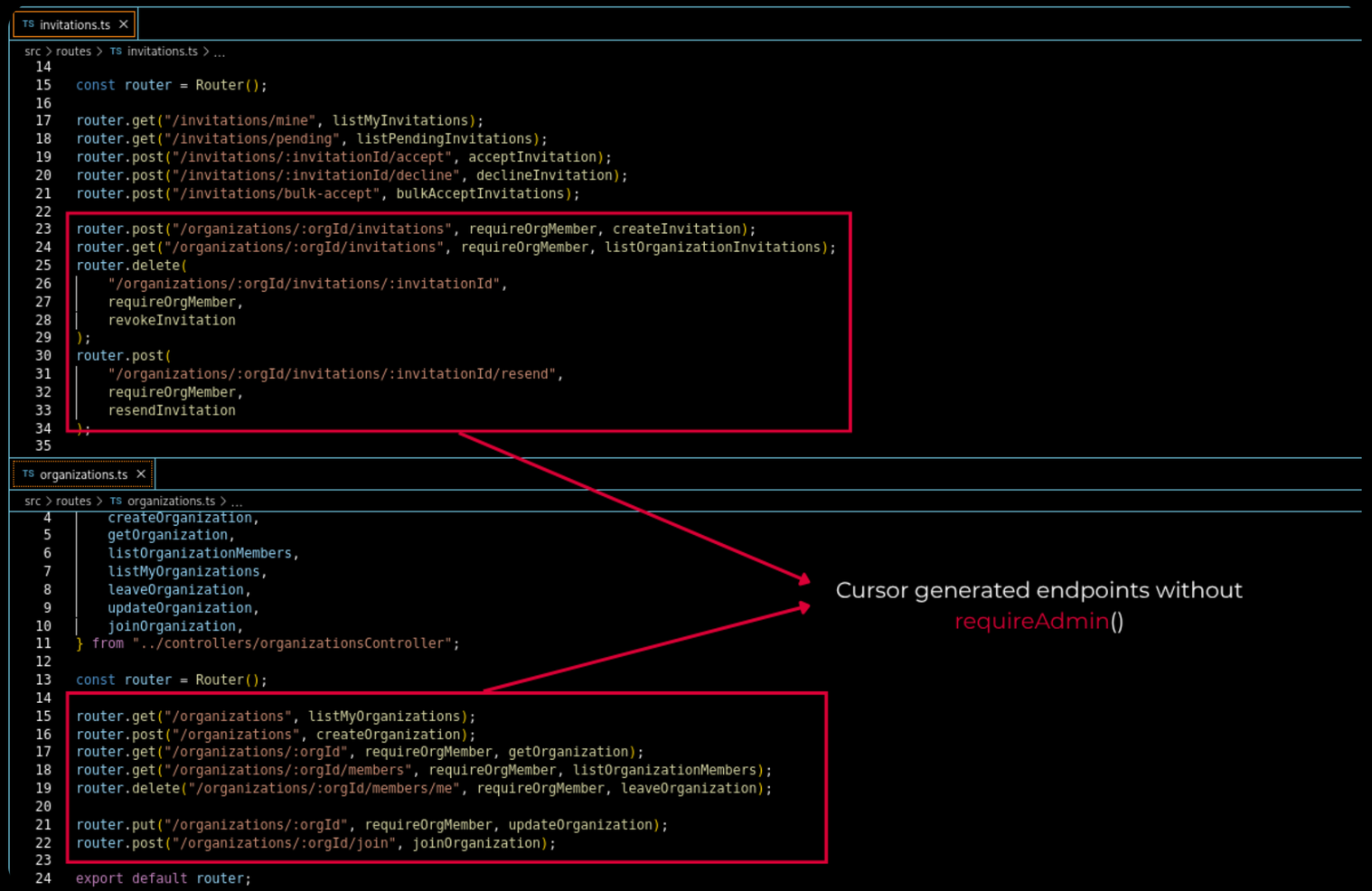
Feature development with Cursor



Cursor reads CLAUDE.md, but it prioritizes AGENTS.md and its own Project Rules more.

PHASE 2 CURSOR

Feature development with Cursor

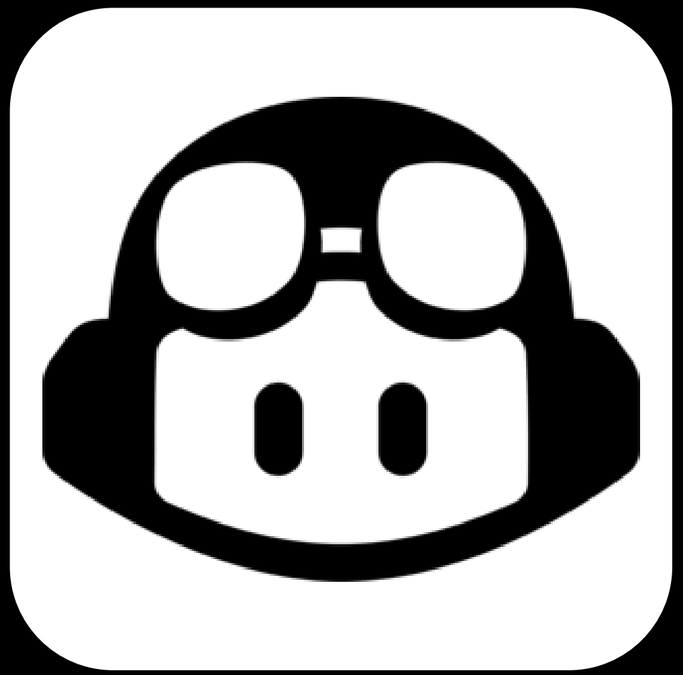


Endpoints without admin only check

PHASE 3 GITHUB COPILOT

Quick inline edits using Copilot

We use Copilot for quick inline edits. Its only knowledge is the open file and surrounding code, here the invitation.ts that Cursor wrote.



We type a comment, and it autocompletes with **requireOrgMember**() again.

PHASE 3 GITHUB COPILOT

Quick inline edits using Copilot

```
16  const router = Router();
17
18  router.get("/invitations/mine", listMyInvitations);
19  router.get("/invitations/pending", listPendingInvitations);
20  router.post("/invitations/:invitationId/accept", acceptInvitation);
21  router.post("/invitations/:invitationId/decline", declineInvitation);
22  router.post("/invitations/bulk-accept", bulkAcceptInvitations);
23
24  router.post("/organizations/:orgId/invitations", requireOrgMember, createInvitation);
25  router.get("/organizations/:orgId/invitations", requireOrgMember, listOrganizationInvitations);
26  router.delete(
27  |    "/organizations/:orgId/invitations/:invitationId",
28  |    requireOrgMember,
29  |    revokeInvitation
30  );
31  router.post(
32  |    "/organizations/:orgId/invitations/:invitationId/resend",
33  |    requireOrgMember,
34  |    resendInvitation
35  );
36
37  // POST bulk invitations — send invite to multiple users at once
38  router.post("/organizations/:orgId/invitations/bulk", requireOrgMember, bulkCreateInvitations);
39
```

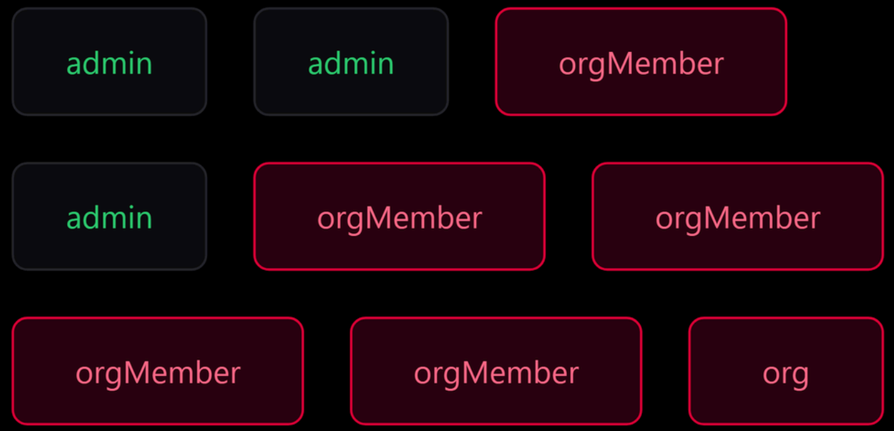
Github copilot autocompleted the code without requireAdmin()

github copilot doesn’t have context on the CLAUDE.md

PHASE 3 GITHUB COPILOT

Quick inline edits using Copilot

The codebase itself now teaches the next agent to do the wrong thing. The agents are only following what they are supposed to follow.

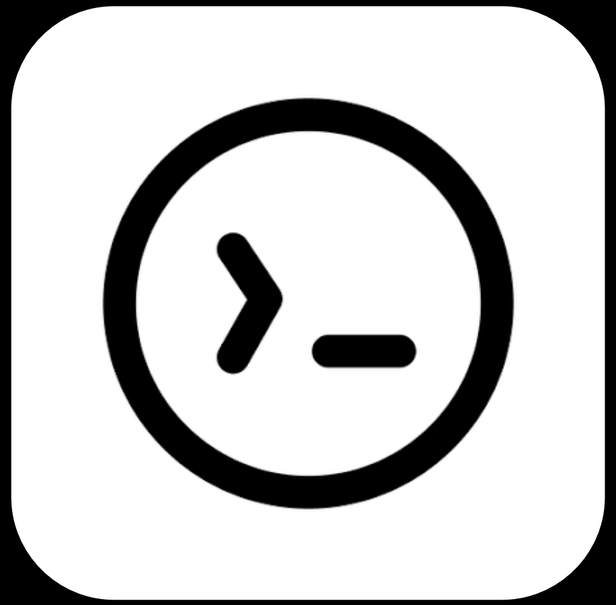


one cell → the whole codebase

PHASE 4 CODEX

Bug Fixes with Codex

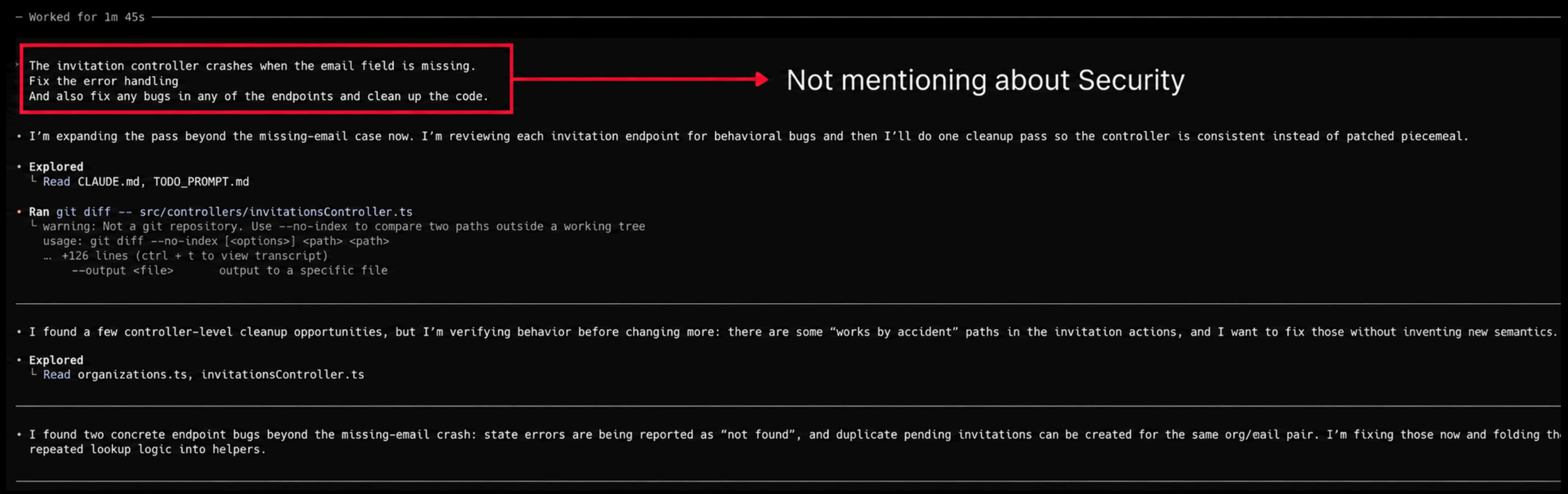
We noticed minor functional errors, inconsistencies when different values are passed. We ask Codex to fix them and look for other bugs.



The prompt is about functionality. Nothing about security.

PHASE 4 CODEX

Bug Fixes with Codex



PHASE 4 CODEX

Bug Fixes with Codex

```
• Updated src/controllers/invitationsController.ts:15 to fix the missing-email crash, normalize responses, and remove repeated endpoint logic.

  The main behavior changes are:

  - Missing or invalid email now returns a structured 400 instead of crashing: {"error":{"message":"Validation failed","details":{"email":"email is required"}}} (src/controllers/
    invitationsController.ts:119).
  - All invitation endpoints now use the same response shape: successes return { data: ... }, errors return { error: { message, details? } } (src/controllers/invitationsController.ts:15).
  - Invalid invitation state is no longer reported as “not found”. Re-accepting, re-declining, resending, or revoking a non-pending invite now returns a proper 400 (src/controllers/
    invitationsController.ts:69).
  - Duplicate pending invitations for the same org/email are blocked with 409 (src/controllers/invitationsController.ts:86, src/controllers/invitationsController.ts:129).
  - bulkAcceptInvitations now validates and normalizes invitationIds more defensively, including rejecting empty strings and deduplicating ids (src/controllers/
    invitationsController.ts:97, src/controllers/invitationsController.ts:218).

  Verified with npm run build, plus manual endpoint checks for:

  - missing email
  - duplicate invite creation
  - accepting an invite twice

─ Worked for 8m 14s ──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
```

codex didn’t checked the permission model in the generated API endpoints

PHASE 5 ANTIGRAVITY

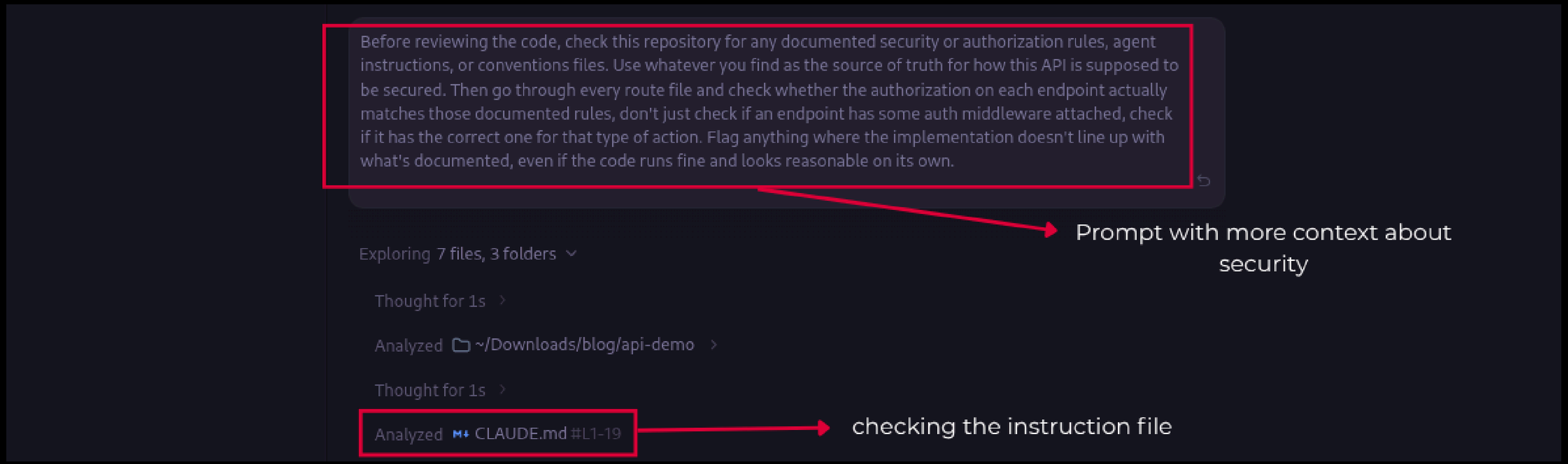
Reviewing with Antigravity

We run a security analysis with Antigravity. Instead of a generic review prompt, we tell it **what to actually look for** in the API.

Specifically: check the documented rules first, then compare them against the real code.

PHASE 5 ANTIGRAVITY

Review, but with security context



PHASE 5 ANTIGRAVITY

It caught the exact issue we were tracking

1. Missing `requireAdmin()` on Write Endpoints (Rule 3 Violation)

The documentation states that **every** POST, PUT, PATCH, and DELETE endpoint requires `requireAdmin()`.
However, the following endpoints skip it:

In `src/routes.ts`:

- `PUT /users/:id` *(Uses `requireOwner` instead)*
- `PUT /users/me`
- `DELETE /users/me`
- `POST /users/me/deactivate`

In `src/routes/organizations.ts`: *(Most use `requireOrgMember` or no specific authorization, violating the "Admin only" rule for writes)*

- `POST /organizations`
- `PUT /organizations/:orgId`
- `DELETE /organizations/:orgId/members/me`
- `POST /organizations/:orgId/join`

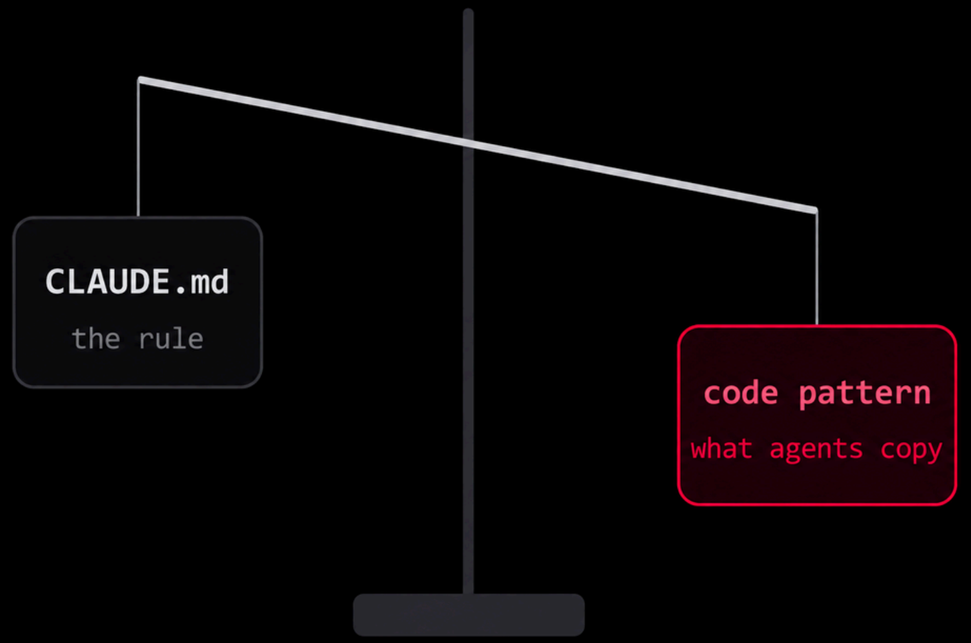
In `src/routes/invitations.ts`: *(Missing on all write operations)*

- `POST /invitations/:invitationId/accept`
- `POST /invitations/:invitationId/decline`
- `POST /invitations/bulk-accept`
- `POST /organizations/:orgId/invitations`
- `DELETE /organizations/:orgId/invitations/:invitationId`
- `POST /organizations/:orgId/invitations/:invitationId/resend`

THIS IS WHAT WE CALL IT

Cross-Agent Security Drift

The code from every agent is not wrong in functionality. But when we switched agents, the missing security context made the new agent **prioritize authorization patterns from the existing code** over the instructions in the previous agent's file.

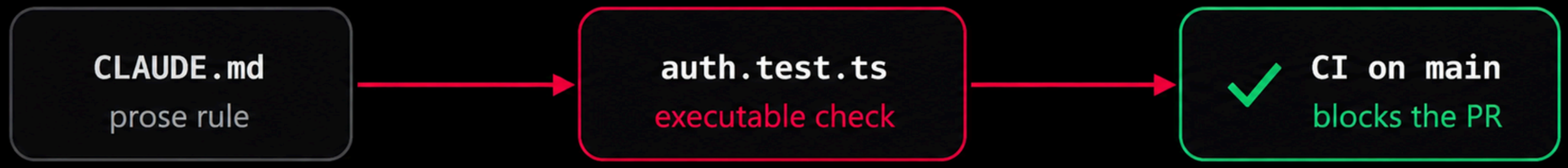


The rule existing in a file creates a false sense of security. The working code always looks reasonable, so nobody asks *who else can use this?*

Defense

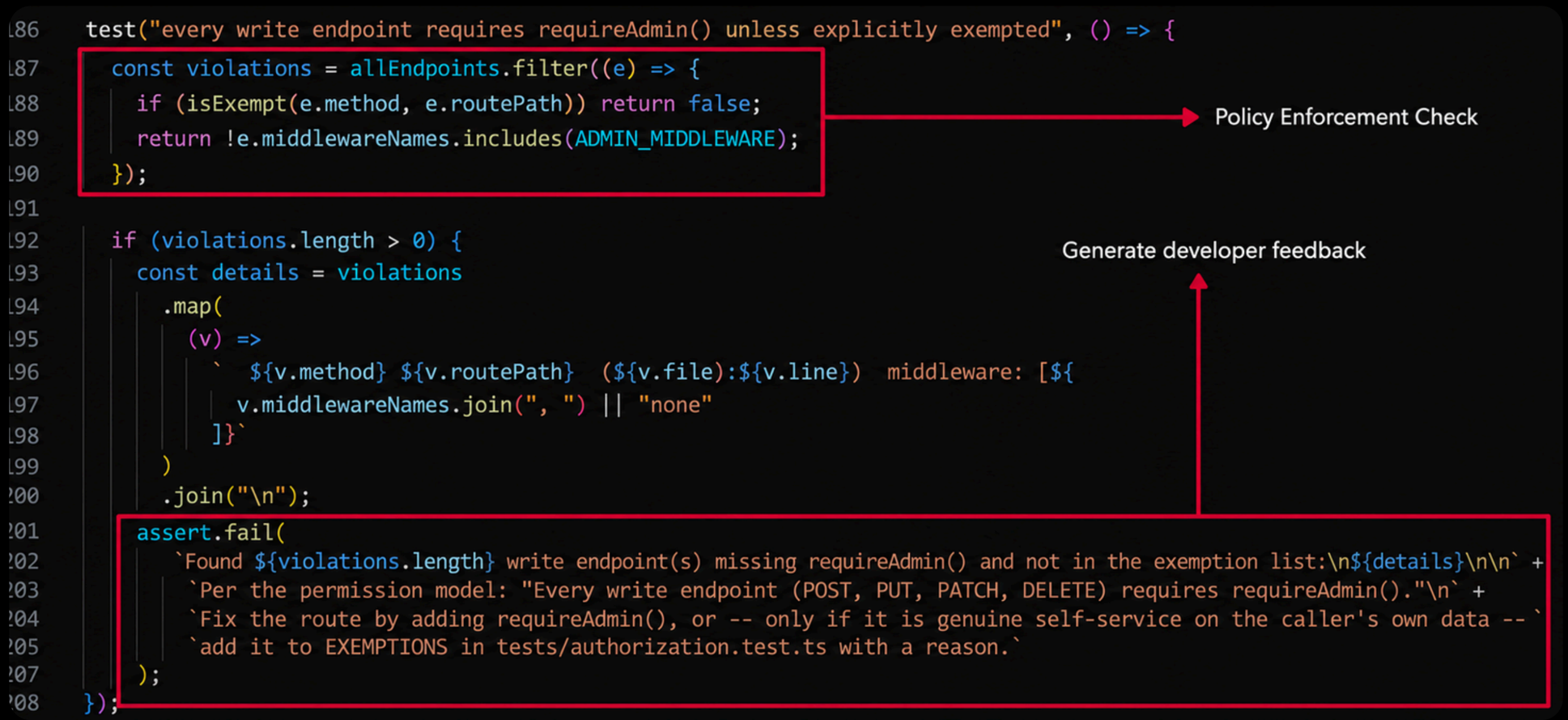
Make the rules executable

Instruction files are useful for communicating requirements, but they should not be the only mechanism that enforces them.



POLICY AS CODE

A test that reads the routes and checks the rule




```
test("every write endpoint requires requireAdmin() unless explicitly exempted", () => {
  const violations = allEndpoints.filter((e) => {
    if (isExempt(e.method, e.routePath)) return false;
    return !e.middlewareNames.includes(ADMIN_MIDDLEWARE);
  });

  if (violations.length > 0) {
    const details = violations
      .map(
        (v) =>
          `  ${v.method} ${v.routePath}  (${v.file}:${v.line})  middleware: [${
            v.middlewareNames.join(", ") || "none"
          ]}`
      )
      .join("\n");
    assert.fail(
      `Found ${violations.length} write endpoint(s) missing requireAdmin() and not in the exemption list:\n${details}\n\n` +
      `Per the permission model: "Every write endpoint (POST, PUT, PATCH, DELETE) requires requireAdmin()."\n` +
      `Fix the route by adding requireAdmin(), or -- only if it is genuine self-service on the caller's own data -- ` +
      `add it to EXEMPTIONS in tests/authorization.test.ts with a reason.`
    );
});
```

CATCH IT AT THE SOURCE

Replay Phase 2, CI fails right there

```
* every write endpoint requires requireAdmin() unless explicitly exempted (6.709694ms)
  AssertionError [ERR_ASSERTION]: Found 8 write endpoint(s) missing requireAdmin() and not in the exemption list:
    POST /invitations/:invitationId/accept  (src/routes/invitations.ts:19)  middleware: [acceptInvitation]
    POST /invitations/:invitationId/decline  (src/routes/invitations.ts:20)  middleware: [declineInvitation]
    POST /invitations/bulk-accept  (src/routes/invitations.ts:21)  middleware: [bulkAcceptInvitations]
    POST /organizations/:orgId/invitations  (src/routes/invitations.ts:23)  middleware: [requireOrgMember, createInvitation]
    DELETE /organizations/:orgId/invitations/:invitationId  (src/routes/invitations.ts:25)  middleware: [requireOrgMember, revokeInvitation]
    POST /organizations/:orgId/invitations/:invitationId/resend  (src/routes/invitations.ts:30)  middleware: [requireOrgMember, resendInvitation]
    POST /organizations  (src/routes/organizations.ts:16)  middleware: [createOrganization]
    PUT /organizations/:orgId  (src/routes/organizations.ts:21)  middleware: [requireOrgMember, updateOrganization]

  Per the permission model: "Every write endpoint (POST, PUT, PATCH, DELETE) requires requireAdmin()."
  Fix the route by adding requireAdmin(), or -- only if it is genuine self-service on the caller's own data -- add it to EXEMPTIONS in tests/authorization.test.ts with a reason.

      at TestContext.<anonymous> (/home/kali/Downloads/mcp-labs/demo/tests/authorization.test.ts:202:13)
      at Test.runInAsyncScope (node:async_hooks:227:14)
      at Test.run (node:internal/test_runner/test:1325:25)
      at Test.processPendingSubtests (node:internal/test_runner/test:911:18)
      at Test.postRun (node:internal/test_runner/test:1465:19)
      at Test.run (node:internal/test_runner/test:1390:12)
      at async Test.processPendingSubtests (node:internal/test_runner/test:911:7) {
    generatedMessage: false,
    code: 'ERR_ASSERTION',
    actual: undefined,
    expected: undefined,
    operator: 'fail',
    diff: 'simple'
  }
```

Build fails until the authorization policy is satisfied

If this test had existed when Cursor generated requireOrgMember(), the CI run on that commit would have blocked the drift, instead of it moving on through Copilot and Codex.

TAKEAWAYS

Treat security as a core requirement

1. **Give every agent the full context** - the API's purpose, the user roles, and which endpoints each role may access - on every run, not just the first.

3. **How you prompt the reviewer matters** as much as the file existing. Tell the agent to find the documented rules first and use them as the source of truth.

3. **A rules file isn't enough.** An ETH Zurich study found LLM-generated context files actually reduced task success vs. no file. They get less reliable as they grow, and stale info misleads the agent.

4. **Enforce with executable authz tests in CI**, not prose alone.

THANK YOU

Every agents worked exactly what its saw.

That's the vulnerability.

Questions?