Agentic Coding: Build

Agentic Cybersecurity — Module 3

Writing production code with an agent

Closing Day 1

What Happens Now

This module and its lab are the last block of Day 1:

1. **Now (M3)**: build ShareBox, a networked file-sharing app, to spec with `small`
2. Your build freezes overnight, exactly as committed
3. **Tomorrow**: attack a summarizer agent (M4), audit and exploit a vulnerable ShareBox (M5), implement the Foundry security spec (M6)

Build exactly to spec, nothing extra — the gap between "passes the tests" and "secure" is tomorrow's lesson.

Workflows That Work

Plan First

Don’t open with “build me an app.” Open with “give me a plan.”

- Have the agent propose: endpoints, data model, order of implementation
- **Review the plan before any code exists** — this is your highest-leverage moment
- Cut everything not in the spec; scope creep is agent nature

A plan review costs two minutes. Un-writing a wrong architecture costs the session.

Small Steps, Verified

The agent's superpower is the feedback loop. Structure the work so the loop has feedback:

- One feature at a time: auth → upload → share links → admin
- Run the relevant tests after each step, not at the end
- Red test → feed the failure back → green test → next feature

Ten small verified steps beat one giant unverified leap. This is how you'll run the ShareBox build in Lab 3.1.

The Build Loop

1 plan → **2 small step** → **3 verify** → **4 commit**

after **4**: back to **2** for the next step ↻ — repeat until the spec is done

small step one reviewable feature or fix — minutes, not hours
verify run the tests / checker now, after this step — not at the end

stuck at **3**, ~3 tries? → extract what was learned → restart fresh → back to **1**

The rest of the module is the two exits: restarts, and what "verify" misses.

When the Agent Gets Stuck

First, recognize the doom loop:

- Same error, third different "fix" → it's guessing
- Diffs getting **bigger** each attempt → more duct tape
- It starts apologizing → it knows, and it'll try again anyway

Rule: **never let it thrash past ~3 attempts** — failures poison the context (Module 1), so attempt four comes out *worse*.

The Operator's Moves

You've called the doom loop. Pick a move:

1. **Stop.** The loop never fixes itself; you are the halting condition
2. **Extract** — fold what the failures taught you into the task statement
3. **Restart** — fresh session, sharper spec, no failure residue
4. **Downgrade** — ask for a smaller step it can actually land
5. **Do it yourself** — hand-write the tricky ten lines, hand the rest back

You practiced extract-and-restart in Lab 2.3; it matters more when the codebase is growing under the agent's hands.

Security of AI-Written Code

What Agent Code Gets Wrong

Agent-written code is **plausible-looking** code — optimized for "works and looks right," not "resists attack." Fast agentic builds reliably produce:

- Missing authorization checks (auth ≠ authz)
- String-built SQL, unsanitized paths, naive file handling
- Secrets in code, verbose error leakage, debug endpoints left in
- Home-rolled crypto and sessions when a library existed

The model isn't bad; **nothing in the loop rewards security**. The tests pass either way.

Why the Tests Pass Either Way

Functional tests check the happy path. Vulnerabilities live off it.

- “Users can download their files” — passes with or without a path traversal
- “Admin endpoint lists users” — passes with or without an authz check
- The agent stops when the signal says done, and the signal is the tests

Tomorrow you’ll work this from the attacker’s side: the ShareBox you audit in M5 passes its whole test suite. It still falls.

What Actually Helps

- Put security requirements **in the prompt, per feature** — "parameterized queries only", "every admin route checks the session role"
- Ask the agent to review its own code for a specific vuln class (decent at finding what it's told to look for; bad at volunteering)
- Use libraries and framework defaults, not hand-rolled mechanisms
- **Independent review** — different session, better model, different eyes (that's M5)

What doesn't: "make it secure" in the prompt. Vague in, vague out.

Model Tiering

Why build with `small` and review with `large`?

- Building is high-volume, feedback-rich: the small model is fast, cheap, good enough when tests verify each step
- Review is judgment-heavy, one-shot: capability gaps show up directly in what gets missed

This is the production pattern too: cheap models where feedback loops catch errors, expensive models where judgment is the product.

Code Review as a Gate

Treat It Like a Junior Engineer's PR

The agent is a prolific junior engineer: fast, confident, zero fear of consequences.

- Its output is a PR, and **you are the reviewer of record**
- Merging unreviewed agent code is merging unreviewed code
- "Tests pass" is where review **starts**, not where it ends

If you can't explain a diff, don't merge it.

What Review Catches That Tests Don't

Tests check behavior. Review checks **judgment**:

- Authorization gaps — the endpoint works, just for everyone
- Injection: string-built SQL, shell interp, unsanitized paths
- Unsafe defaults: debug on, permissive CORS, world-readable uploads
- Secrets in code, keys in config, tokens in logs

Every one passes the functional suite — you saw this list two slides ago.

Reviewing With an Agent

Agents can staff the review, if you set it up honestly:

- **Second model, second session** — never the one that wrote it: same blind spots, and it defends its own work
- **Checklists beat "find bugs"** — "check every route for authz", "list every string-built query"
- The reviewer flags; **you** decide what merges

Tomorrow, M4 shows how a reviewing agent can itself be attacked, and in M5 this workflow **becomes the audit workflow**, pointed at a deliberately vulnerable ShareBox.

Accountability

You Own What Ships

The agent is a power tool, not a colleague. When agent-written code has a vuln in production:

- "The AI wrote it" is not a defense — your name is on the commit
- Review discipline doesn't relax because generation got fast; **it tightens**, because volume went up

Corollary for this class: the ShareBox you ship this hour is **yours**, and it freezes tonight as-is. Review it before it does.

Wrap-Up

What You Should Take Away

- Plan first; review the plan, not just the code
- Small verified steps; feed failures back; ~3 failed attempts ⟶ extract and restart
- Functional tests cannot see vulnerabilities — the happy path lies
- Security requirements go in the prompt, specifically, per feature
- Agent code is a junior's PR: review is the gate, you're the reviewer of record
- Tier your models: cheap to build, strong to judge
- You own what the agent ships

Lab 3.1: build ShareBox to spec with `small` — the last lab of Day 1; it freezes overnight