Certified Secure AI Developer

Curriculum - 2026

This training is built for developers, not for the security team auditing them afterward. Every module runs against a live coding-agent environment your team already recognizes, not a slide deck about AI risk. By the end of the day, developers can configure agent guardrails, write specs that carry their own security acceptance criteria, contain secrets and blast radius, and wire policy checks directly into the agent loop, then prove it under a proctored exam.

D1 — Agentic Coding Foundations & Threat Modeling

Trace how a coding agent turns a tool call into a file write or shell command, map its reach across the working tree, git history, and CI, and threat-model the ways that reach gets abused.

Topics covered

- How coding agents turn a tool call into a file write, shell command, or pull request
- Mapping agent reach: working tree, git history, CI/CD pipelines, package registries
- Threat-modeling the agent's trust boundary: indirect prompt injection via repo content
- Slopsquatting and hallucinated or AI-suggested dependency risk
- Autonomy escalation: when an agent grants itself more permission than intended

LAB: Compromise Your Own Agent Plant an indirect-injection payload in a realistic repo artifact (a README, an issue comment, a code comment), run an ordinary agent task, and trace exactly where the trust boundary failed.

D2 — Spec-Driven Development as a Security Control

Write specs with abuse cases and executable acceptance criteria — the control most AI-adoption programs skip entirely.

Topics covered

- Why specs, not code review, are the first real security control in an agentic workflow
- Writing abuse cases into feature specs (misuse-case driven spec authoring)
- Executable acceptance criteria: turning a security requirement into a testable condition
- Spec review patterns that catch under-specified authorization and validation logic

LAB: Abuse-Case Spec Sprint Given a feature ticket, write a spec with abuse cases and acceptance criteria, run it through the agent, and grade whether the resulting code actually satisfies the security criteria.

D3 — Encoding Standards & Paved Roads

Build the rules files and paved roads an agent actually follows, then adversarially test them until they hold.

Topics covered

- What a "paved road" is, and why an agent needs one it can't wander off
- Writing agent rules files and project conventions that encode secure defaults
- Adversarially testing your own paved road with prompts designed to route around it
- Versioning and maintaining rules files over time

LAB: Break Your Own Rules File Write a rules/paved-road file for a sample repo, red-team it with adversarial prompts until it holds or breaks, then patch the gap you find.

D4 — Secrets, Permissions & Blast Radius

Close the leak paths agents open by default: permission models, sandboxing, worktree containment, and brokered short-lived credentials in place of secrets sitting in the repo.

Topics covered

- Default leak paths: how agents end up with secrets in plaintext, logs, or context
- Permission models and sandboxing for agent tool calls
- Worktree containment: limiting what an agent can touch outside its task scope
- Brokered, short-lived credentials vs. static keys sitting in the repo

LAB: Contain the Blast Radius Reconfigure a deliberately over-permissioned agent environment to use short-lived brokered credentials and sandboxed tool access, then verify the agent can no longer reach out-of-scope resources.

D5 — Hooks & Policy-as-Code in the Agent Loop

Wire pre-execution and pre-commit gates directly into the agent loop, with remediation loops and tamper resistance — not a policy doc nobody reads.

Topics covered

- Pre-execution vs. pre-commit gates: where to intercept an agent's actions
- Writing policy-as-code checks that run inside the agent loop
- Remediation loops: failing safely and feeding the failure back to the agent
- Tamper resistance: keeping an agent from disabling its own guardrails

LAB: Wire the Gate Implement a pre-commit policy hook that blocks a class of insecure change, confirm the agent can't bypass or disable it, and test the remediation loop end-to-end.

D6 — AI Code Security in the SDLC & Rollout

Run agent-time and CI-time scanning, enforce dependency provenance and AI-authored PR review discipline, and leave with a 90-day rollout blueprint — risk tiers, org defaults, approved registries.

Topics covered

- Agent-time vs. CI-time scanning: where each catches what the other misses
- Dependency provenance and verifying AI-suggested packages before merge
- AI-authored PR review discipline: what changes for a reviewer vs. human-authored code
- Building a 90-day rollout blueprint: risk tiers, org defaults, approved registries

LAB: Ship the Rollout Blueprint Using your own org's stack as the working example, draft a 90-day rollout plan — risk tiers, default agent permissions, approved package/tool registries — that a security leader could hand to engineering tomorrow.

CERTIFICATION EXAM DETAILS

Part 1 - Knowledge & Judgment Exam

Scenario-based questions tied to a specific environment or proctored artifact — not lookup trivia.

Part 2 · Challenge Exam

Four to six graded challenges in fresh, randomized, browser-based lab environments, auto-graded against a verifiable end state.

Part 3 · Proctored Capstone + Live Defense

One continuous proctored sitting. Candidates secure an agentic-development workflow for a supplied codebase — spec, abuse cases, rules pack, hooks, sandbox/permission config — then defend the decisions live.

To pass, candidates must clear a minimum threshold in every instrument — no averaging a strong knowledge score past a weak capstone — plus a pass on the live defense.

Certification is valid for 24 months. Candidates leave with a verifiable digital badge for LinkedIn.