Scope - Understanding The Rubric

Scope Description
Design and implement an AI-assisted secure prompt and code-specification checker for developers using AI code generators. The system will take a natural-language prompt before it is sent to an AI coding tool, analyze it for missing or weak security requirements, flag malicious prompts, and produce a rewritten, security-hardened prompt. The tool should support a simple developer-facing interface, integrate with at least one LLM, and demonstrate clear improvements in security coverage, innovation, and usability as defined in the hackathon rubric.

Acceptance Criteria

- The system analyzes a prompt for common vulnerability classes (e.g., injection, auth/session issues, crypto misuse, secrets handling, insecure defaults) and clearly flags any risks it detects.
- For each flagged risk, the tool either: rewrites the prompt to include appropriate security requirements, or explains why the prompt is unsafe and refuses to proceed.
- Uses at least one LLM-based component in a non-trivial way (e.g., “blue-team” prompt hardener, optional “red-team” attacker that proposes exploits, etc.).
- Demonstrates a clear pre-AI-generation workflow: prompt in → analysis → secure rewrite (and optional attack simulation) → final prompt for external AI code generator.
- Includes at least one design artifact showing the prompt flow and AI components (e.g., architecture diagram, sequence/flow diagram, or UML).
- Provides a simple, clear interface (CLI or small web front end) where a developer can:
    1. Enter prompt
    2. see findings and improved prompt (if not rejected for malicious use)
    3. Iterate / make changes / tradeoffs
    4. Code preview and export option
    5. See prompting history
- Response format is structured and readable (e.g., sections for “Issues found”, “Rewritten prompt”, “Security notes”).
- A 5-minute demo can show end-to-end usage for at least two developer personas (e.g., junior dev and security engineer).
- A working MVP is delivered that runs on sample prompts without crashing.
- Work is distributed by skillset or learning goals, with evidence of small merges and iterative builds (not a single big-bang push).

Deliverables

- Requirements Artifacts (3+ total across categories)
    1. Scope documentation
    2. Flow diagrams (front and back end)
    3. End to end testing documentation
- Implementation: Running MVP (CLI or web front end) that demonstrates prompt analysis and secure rewrite.
- Demo script and 5-minute presentation deck aligned with the rubric (Planning, Development, Presentation).

Project Exclusions

- Full static code analysis or full repo scanning (tool operates on prompts/specs, not entire codebases).
- Guarantees of “perfectly secure” code; the tool is a pre-check assistant, not a formal verification engine. It does its best though
- Deep integration with commercial AI platforms beyond simple “copy/paste” or a basic API call, if used.

Constraints

- Implemented primarily in technologies the team already uses (e.g., JavaScript/TypeScript + React, or Python/Node + CLI) within the hackathon timeframe.
- Must be demo-able on a single laptop
- Hackathon timebox: all features, artifacts, and demos must be ready by judging deadline.

Assumptions

- Target users are developers who already understand basic software security concepts and regularly use AI-assisted code generation.
- Evaluation may use sample prompts in a small set of languages (e.g., JavaScript, Python, Java), not every possible tech stack.
- Internet access and at least one LLM endpoint (hosted or local) are available during development and demo.
- No major pivot in problem statement (focus remains on secure code generation via better prompts/specs, not general-purpose IDE tooling).