Secure OTA Update Compiler

14-Week Execution Plan

Phase 1: Problem Understanding & Basic Foundations

Week 1 – Problem & Context Understanding

Activities

- Understand firmware update (OTA) workflow at a conceptual level
- Identify security risks in firmware update logic
- Clearly define what the compiler is responsible for vs what it is not

Deliverables

- Problem definition document
- Scope boundaries (what is enforced at compile time)
- Initial threat overview (malicious firmware, rollback attack)

Week 2 – Literature Survey & Gap Identification

Activities

- Review 10–15 sources on:
    - Secure firmware updates
    - Compiler-based security enforcement
    - Static enforcement of security invariants
- Study limitations of runtime-only update checks

Deliverables

- Literature review table (Approach | Technique | Limitations)
- Gap statement

Phase 2: Requirement Analysis & System Design

Week 3 – Requirements & Threat Analysis

Activities

- Define functional requirements
- Define non-functional requirements
- Create a threat model focused on update abuse

Deliverables

- Software Requirements Specification (SRS)
- Threat model document (attacker goals & assumptions)

Week 4 – Architecture & Design Planning

Activities

- Design overall compiler architecture:
  - Input → AST → CFG → Security Checks → Output
- Identify compiler modules:
  - Update detector
  - Control-flow analyzer
  - Invariant checker
- Choose tools (LLVM / Clang or custom compiler)

Deliverables

- Architecture diagram
- Module interaction diagram
- Technology stack justification

Phase 3: Language and Front-End Setup

Week 5 – Update Logic Identification Design

Activities

- Define what constitutes "firmware update code"
- Specify expected APIs/functions (e.g., verify, install)
- Define assumptions about firmware structure

Deliverables

- Update logic specification
- Sample secure vs insecure firmware snippets
- Defined enforcement rules (informal)

Week 6 – Frontend / Parsing / AST Access

Activities

- Implement parsing frontend (LLVM AST pass or custom parser)
- Identify firmware update functions in AST

Deliverables

- Identifiable IR generation

- Logs showing detected update-related code
- AST traversal module

Phase 4: Core Compiler Analysis

Week 7 – Control Flow Graph (CFG) Construction

Activities

- Construct CFG for update-related functions
- Identify execution paths leading to firmware installation
- Understand dominance and reachability

Deliverables

- CFG diagrams
- CFG construction documentation
- Intermediate analysis logs

Week 8 – Security Invariant Definition

Activities

- Formally define the four enforced invariants:
    1. Signature verification before installation
    2. Firmware version monotonicity (rollback prevention)
    3. Trusted update source validation
    4. Information Leakage Through Debug Logs

Deliverables

- Formal invariant definitions
- Rule-checking logic description
- Error conditions for each violation

Phase 5: Security Enforcement Implementation

Week 9 – Compile-Time Enforcement Logic

Activities

- Implement checks for:
    - Verify → Install dominance
    - Version comparison enforcement
    - Trusted source usage
    - No log leakage
- Ensure violations trigger compilation failure

Deliverables

- Working enforcement engine
- Compile-time error messages
- Rule enforcement logs

Week 10 – Instrumentation

Activities

- Implement safe-code instrumentation:
    - Auto-insert version checks
    - Insert mandatory verification stubs
- Log enforcement decisions for auditability

Deliverables

- Instrumented firmware output
- Before/after code comparison
- Instrumentation documentation

Phase 6: Testing & Validation

Week 11 – Testing & Validation

Activities

- Develop test cases:
    - Unsigned firmware
    - Rollback attempt
    - Valid secure update
- Perform compile-time validation

Deliverables

- Test case suite
- Test results and failure analysis
- Bug-fix documentation

Week 12 – Evaluation & Demonstration Setup

Activities

- Compare normal compilation vs secure compilation
- Measure detection coverage and enforcement success
- Prepare VM or Raspberry Pi demo environment

Deliverables

- Evaluation report
- Demo screenshots or logs
- Comparative analysis

Phase 7: Explanation, Documentation & Refinement

Week 13 – Explainability & Documentation

Activities

- Document how each invariant is enforced
- Explain compiler reasoning with CFG examples
- Prepare diagrams and flow explanations

Deliverables

- Explanation report
- CFG-based enforcement illustrations
- Draft final report

Week 14 – Final Integration & Submission

Activities

- Final code cleanup and refactoring
- End-to-end integration testing
- Prepare presentation and demo

Deliverables

- Final compiler implementation
- Final report (PDF)
- Presentation slides
- Demo video / live demo