Technology Stack Justification

Dhruv Sunil Bhatia
24CSB0A19
CSE - A
Q119 - Secure OTA Update Compiler

---

1 Overview

This section justifies the selection of technologies used for implementing the Secure OTA Update Compiler, focusing on static analysis capability, extensibility, and suitability for compile-time security enforcement.

2 Target Firmware Language (C / C++)

C and C++ are widely used in embedded and IoT firmware development. Targeting these languages ensures compatibility with real-world OTA update logic, direct access to low-level system APIs, and alignment with existing firmware ecosystems.

3 Compiler Infrastructure (LLVM / Clang)

LLVM and Clang provide a modular and extensible compiler architecture with rich access to Abstract Syntax Trees (AST) and Control Flow Graphs (CFG). This makes them well-suited for implementing compile-time security enforcement mechanisms.

4 Static Analysis Framework (LLVM Analysis Passes)

LLVM analysis passes support precise CFG construction, dominance analysis, and path-sensitive checks. These capabilities are essential for verifying that mandatory security checks occur on all execution paths leading to firmware installation.

5 Security Enforcement Logic (Custom Compiler Passes)

Custom compiler passes encode OTA firmware security invariants as compile-time correctness rules. This ensures that insecure firmware update logic is rejected during compilation rather than discovered at runtime.

6 Instrumentation Support (AST / IR Rewriting)

Optional instrumentation allows the compiler to safely insert mandatory security checks using AST or Intermediate Representation (IR) rewriting techniques. This enables compile-time hardening without modifying source code manually.

7 Build and Integration Environment (LLVM Toolchain)

The standard LLVM toolchain enables seamless integration of the Secure OTA Update Compiler into existing firmware build pipelines, minimizing adoption effort for developers.

8 Component-wise Technology Justification

Component	**Technology**	**Justification**
Firmware Language	C / C++	Dominant languages for embedded firmware
Compiler Framework	LLVM / Clang	Extensible with AST and CFG access
Analysis Engine	LLVM Passes	Control-flow aware static analysis
Security Enforcement	Custom Passes	Domain-specific security invariants
Instrumentation	AST / IR Rewriting	Safe compile-time hardening
Build System	LLVM Toolchain	Easy integration into existing workflows