Software Requirements Specification (SRS)

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

---

1 Introduction

1.1 Purpose

This document specifies the functional and non-functional requirements for the Secure OTA Update Compiler. The compiler statically analyzes firmware update logic and enforces mandatory security invariants at compile time. Any firmware update code that violates these invariants is rejected, preventing insecure firmware from being generated or deployed.

1.2 Scope

The system operates as a compiler extension or static analysis pass integrated into the firmware build process. It focuses exclusively on firmware update logic, ensuring that critical security checks are present and correctly ordered along all execution paths leading to firmware installation.

The compiler does not:

- Perform runtime enforcement
- Manage key distribution or firmware servers
- Replace secure boot or hardware trust anchors

1.3 Definitions and Acronyms

- OTA: Over-The-Air firmware update
- Invariant: Mandatory security property that must hold on all valid execution paths
- CFG: Control Flow Graph
- AST: Abstract Syntax Tree

2 Overall Description

2.1 Product Perspective

The Secure OTA Update Compiler is positioned between source code and firmware binary generation. It analyzes source-level update logic using AST and CFG representations before emitting compiled firmware.

High-Level Flow:
Input Source Code → Parsing → AST → CFG → Security Invariant Checks → Accept / Reject → Firmware Binary

2.2 User Classes and Characteristics

- **Firmware Developers**: Write update logic and receive compile-time errors on violations
- **Security Auditors**: Review compiler logs and invariant enforcement reports
- **System Integrators**: Integrate the compiler into build pipelines

2.3 Operating Environment

- Host OS: Linux
- Target: Embedded / IoT firmware
- Compiler Framework: LLVM/Clang or equivalent static analysis framework

2.4 Assumptions and Dependencies

- Firmware update logic is written in C/C++ or a supported language
- Update installation occurs through identifiable function calls
- Cryptographic verification APIs are statically identifiable

3 Functional Requirements

- **FR-1: Firmware Update Logic Detection**
  The compiler shall identify firmware update–related functions, including verification, version checks, and installation routines.
- **FR-2: Control Flow Analysis**
  The compiler shall construct a CFG for all update-related functions and identify all execution paths leading to firmware installation.

- **FR-3: Signature Verification Enforcement**
  The compiler shall ensure that cryptographic signature verification dominates firmware installation on all valid paths.
- **FR-4: Rollback Protection Enforcement**
  The compiler shall verify firmware version monotonicity checks prior to installation.
- **FR-5: Trusted Source Validation**
  The compiler shall ensure that firmware originates from a trusted and authenticated source.
- **FR-6: Sensitive Information Leakage Prevention**
  The compiler shall detect and reject update logic that logs or exposes sensitive firmware or cryptographic data.
- **FR-7: Cryptographic Policy Enforcement**
  The compiler shall reject usage of weak or disallowed cryptographic primitives within update logic.
- **FR-8: Compile-Time Rejection**
  The compiler shall terminate compilation with explicit error messages if any security invariant is violated.
- **FR-9: Enforcement Reporting**
  The compiler shall generate logs detailing enforced invariants and detected violations.

4 Non-Functional Requirements

- **NFR-1: Security** – Guaranteed sound enforcement
- **NFR-2: Performance** – Acceptable compilation overhead
- **NFR-3: Usability** – Precise and actionable error messages
- **NFR-4: Extensibility** – Easy addition of new invariants
- **NFR-5: Reliability** – Deterministic compilation results

5 External Interface Requirements

5.1 Compiler Interface

- Input: Firmware source code
- Output: Firmware binary or compilation error report

5.2 Logging Interface

Text-based logs describing invariant checks and violations.

6 Constraints

- Static analysis only (no runtime instrumentation)
- Language and framework constraints of the compiler infrastructure
- Limited visibility into hardware-enforced security mechanisms

7 Acceptance Criteria

The system is considered complete when:

- All defined invariants are enforced on all control-flow paths
- Insecure firmware update samples fail compilation
- Secure firmware update samples compile successfully

8 Future Enhancements

- Automated code instrumentation for compliant fixes
- Support for additional programming languages
- Integration with CI/CD pipelines

Threat Model Document

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

---

1 Introduction

1.1 Purpose

This document defines the threat model for the Secure OTA Update Compiler. It identifies potential attackers, their capabilities and goals, key assets, trust assumptions, and security threats related to firmware over-the-air (OTA) update mechanisms. The threat model guides the definition and enforcement of compile-time security invariants.

1.2 Scope

The threat model focuses exclusively on firmware update logic implemented in software and analyzed at compile time. It does not cover hardware-level attacks, physical tampering, or post-deployment operational security beyond firmware installation.

2 System Overview

The system under analysis is a compiler-based static analysis and enforcement mechanism that inspects firmware update code before binary generation.

Analyzed Components:

- Firmware update functions (verify, download, install)
- Control-flow paths leading to firmware installation
- Cryptographic API usage within update logic
- Logging and debug statements related to updates

Out of Scope:

- Secure boot enforcement
- Key provisioning and management
- Network-layer protections
- Runtime monitoring

3 Assets

The following assets must be protected:

- Firmware integrity
- Firmware authenticity
- Version correctness
- Confidential update data
- System availability

4 Threat Actors

4.1 Remote Network Attacker

Capabilities: Intercept, modify, or replay firmware update payloads
Limitations: No physical device access

4.2 Malicious Firmware Provider

Capabilities: Craft malicious firmware images
Limitations: Cannot break strong cryptography

4.3 Compromised Update Server

Capabilities: Serve attacker-controlled firmware
Limitations: Cannot bypass compiler enforcement

4.4 Insider / Developer Error

Capabilities: Introduce insecure or incomplete update logic unintentionally
Limitations: No malicious intent

5 Attacker Goals

- Install malicious firmware
- Downgrade firmware to vulnerable versions
- Bypass signature or integrity checks
- Inject unauthorized update sources
- Extract sensitive data via logs

- Cause denial of service

6 Threat Enumeration

- **T1: Malicious Firmware Installation** – Installation without cryptographic verification
- **T2: Rollback Attack** – Forced installation of older vulnerable firmware
- **T3: Untrusted Update Source** – Acceptance of unauthenticated firmware
- **T4: Information Leakage via Logs** – Exposure of sensitive update data
- **T5: Weak Cryptographic Usage** – Use of deprecated or insecure crypto primitives

7 Threat Mitigations (Compiler-Enforced)

Threat ID	**Mitigation Strategy**
T1	Enforce signature verification dominance before installation
T2	Enforce firmware version monotonicity checks
T3	Enforce trusted source validation APIs
T4	Reject sensitive logging statements in update paths
T5	Enforce cryptographic policy at compile time

All mitigations are applied statically, and violations result in compilation failure.

8 Trust Assumptions

- The compiler is trusted and uncompromised
- Cryptographic primitives are secure when used correctly
- Secure boot mechanisms (if present) operate correctly
- Identified update APIs correctly represent firmware installation

9 Security Goals

The Secure OTA Update Compiler guarantees that:

- No firmware installation code exists without mandatory security checks
- All update-related execution paths satisfy defined invariants

- Insecure update logic is eliminated before deployment

10 Residual Risks

- Logic errors outside recognized update code
- Hardware-level or physical attacks
- Runtime vulnerabilities unrelated to update logic

These risks are explicitly outside the compiler's enforcement scope.

11 Conclusion

This threat model demonstrates that critical OTA update threats often arise from missing or incorrectly ordered software logic rather than cryptographic failure. By enforcing update security invariants at compile time, the Secure OTA Update Compiler mitigates high-impact attack classes before firmware deployment.