CFG Construction Documentation

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

---

1 Introduction to Control Flow Graph (CFG)

A Control Flow Graph (CFG) is a directed graph representation of all possible execution paths within a function. In LLVM, each function consists of BasicBlocks. Each BasicBlock contains a sequence of instructions ending with a terminator instruction (such as a branch or return). The edges in the CFG are determined by these terminator instructions.

2 CFG Construction in LLVM

The LLVM Intermediate Representation (IR) generated from C source code was analyzed using a custom LLVM Function Pass. The pass traverses each BasicBlock in the `updateFirmware()` function and prints successor relationships using LLVM's built-in CFG utilities.

Additionally, the `-dot-cfg` option of `opt` was used to automatically generate CFG visualization files in DOT format for graphical analysis.

3 Analysis of `insecure.c`

In the insecure implementation, the `updateFirmware()` function directly calls `install()` without performing signature verification, rollback checks, or source validation.

The generated CFG diagram (`insecure_cfg.pdf`) shows a linear control flow structure where execution moves directly from entry to installation and then to return, without intermediate validation branches.

4 Analysis of `secure.c`

In the secure implementation, multiple conditional branches are introduced before the `install()` call. These include:

- Signature verification
- Rollback protection (version comparison)
- Trusted source validation

The CFG diagram (`secure_cfg.pdf`) shows multiple branching BasicBlocks that must be traversed before reaching the installation block, demonstrating guarded execution.

5 Execution Path Identification

Using CFG analysis, the BasicBlock containing the `install()` call was identified.

- In the insecure case, the installation block is directly reachable from the entry block.
- In the secure case, all execution paths reaching `install()` must pass through validation checks.

6 Dominance and Reachability

Dominance analysis was performed using LLVM's `DominatorTree`. A BasicBlock A dominates block B if every path from the function entry to B passes through A.

In the secure implementation, validation checks dominate the installation block. In contrast, in the insecure implementation, no such dominance relationship exists between validation logic and installation.

7 Conclusion

Week 7 successfully established structural control flow analysis of firmware update logic using LLVM IR. The generated CFG diagrams and analysis logs demonstrate clear structural differences between secure and insecure update implementations, forming the foundation for compile-time enforcement of security invariants.