Fie on Firmware!

Finding Vulnerabilities in Embedded Systems using Symbolic Execution

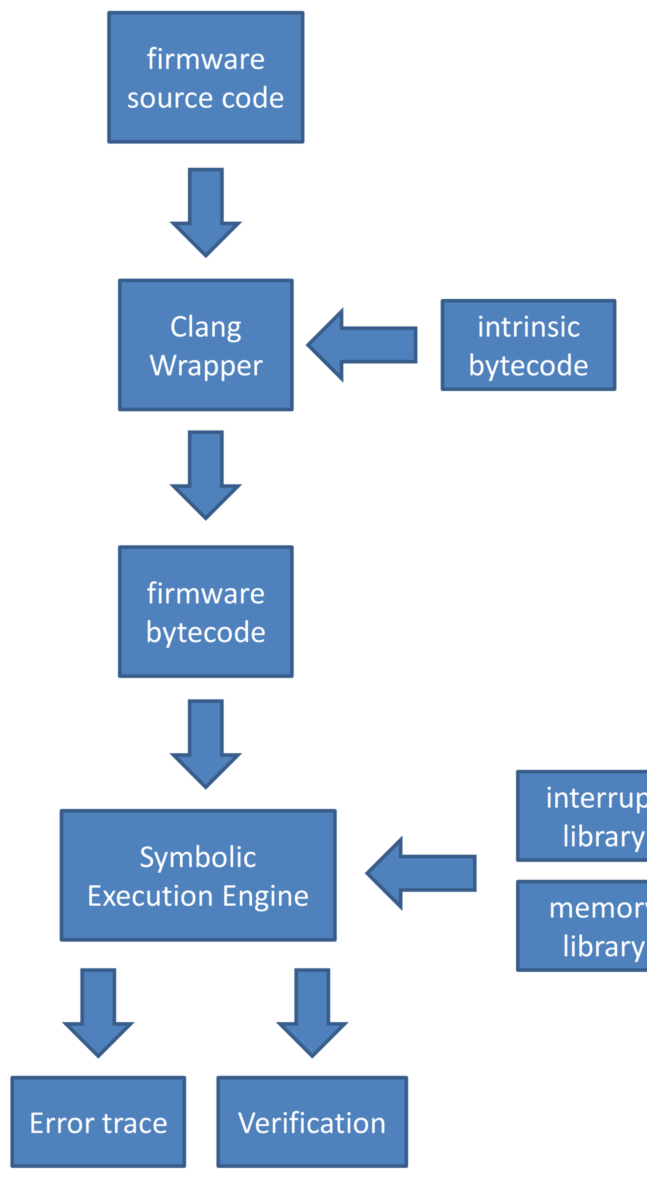
Drew Davidson, Benjamin Moench, Somesh Jha, Thomas Ristenpart

Problem
- Embedded devices ubiquitous, and security critical
- Highly diverse architectures and varied deployments
- Many memory vulnerabilities reported in the wild

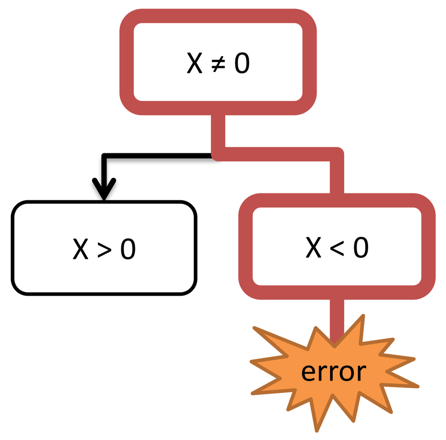
Basic Approach
- Heavyweight program analysis for lightweight firmware
- Modular design:
  - Pluggable architecture model
  - Pluggable interrupt model
- Verification in many cases

Key Payoffs
- Flexible analysis fidelity levels
  - Bugfinding for larger firmwares
  - Verification for smaller firmwares
- Can handle many different deployments
  - Handles over 150 MSP430 models

FiE Workflow



Symbolic Execution Techniques
- Symbolic execution based on KLEE
  - Allows memory locations to be treated as a system of constraints
  - SAT solver explores all feasible program paths
- Most symbolic execution engines maintain the frontier of explored memory states
- FiE maintains entire tree of program states

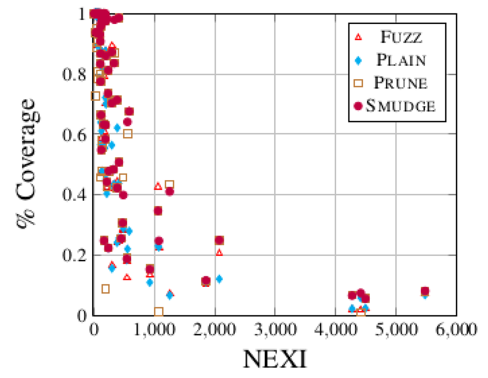


Evaluation
- Corpus of 99 MSP430 firmwares
  - 12 TI community
  - 1 Synthetic
  - 8 USB protocol stack
  - 78 Github
- Ran tests for 50 minutes on Amazon EC2
  - 16-bit KLEE
  - FiE (+pruning) (+smudging)

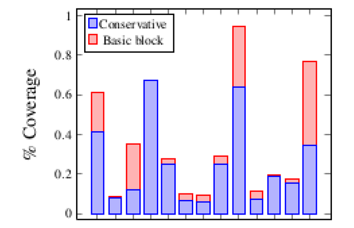
Results

Mode	Termination Status			FPs
	No mem	Timeout	Finished	
Base	9	2	88	93
Fuzz	10	79	10	0
Plain	7	85	7	0
Prune	0	64	35	0
Smudge	0	46	53	1

Bugfinding / Validiation: This table shows the effectiveness of FiE over Base (unmodified version of KLEE), Fuzz (Fuzz testing memory model), Plain (FiE with no optimizations), Prune (FiE with pruning only), and Smudge (FiE with both pruinging and smudging. FiE is a marked improvement over KLEE, and has a low False Positive (FP) rate.



Coverage Results: Percentage of firmware covered by FiE, sorted by number of executable instructions (NEXI). These results show that pruning and smudging are both effective optimizations, with Smudging being the most effective



Relaxed Interrupt Model: Most analysis time is spent in interrupts. Here, the coverage is shown for the 13 most challenging firmwares (in terms of exhaustive program coveage). Relaxing the interrupts to fire once per basic block can have dramatic coverage gains

Conclusions
- Static analysis is a good fit for embedded environments
- Traditionally out-of-reach analysis goals often achievable

Future Work
- Wider variety of analysis targets
  - Embedded Operating Systems
  - Wider variety of chipsets
- Push heavyweight symbolic execution back to desktop applications

Modular Memory Library
- Specify single file that lays out memory ranges and offets
- Special memory semantics:
  - RAM – standard semantics
  - Flash – must be unlocked to write
  - Peripheral – user-defined semantics
- Supports diverse analysis
  - Symbolic execution
  - Fuzzing

Modular Interrupt Library
- Specify frequency of interrupts
  - Timers
  - Peripheral events
- Most execution time is spent in interrupts

Optimizations
- Pruning
  - Maintain previously seen states
  - Discard paths with identical state
- Smudging
  - Replace volatile concrete states with symbolic over-approximation

Learn More
- Fie Presentation:

  http://bit.ly/1lHzE0V
- Download FiE:

  http://bit.ly/1n5W9Pg