FiE on Firmware
Finding Vulnerabilities in Embedded Systems using Symbolic Execution

Drew Davidson
Ben Moench
Somesh Jha
Thomas Ristenpart
FiE in a Nutshell

• Symbolic execution tailored to embedded firmware
  – Detects common firmware vulnerabilities
  – Deals with domain-specific challenges
  – Able to verify small programs

• Tested on 99 programs
  – Found 22 bugs
  – Verified memory safety for 52 programs
Example Attack: WOOT 2012

16-bit low power device
C firmware
Low-level hardware interaction

Buffer Overflow!

[Frisky et al., 2012]
Source code analysis is helpful on desktop

Could be transitioned to firmware

... Little Work on Detecting Vulnerabilities
Symbolic Execution

- Represents program input as sets of constraints
- Explores multiple feasible paths for bugs
- Provide detailed trace to vulnerability
- KLEE
  - Popular, mature tool
  - Average > 90% line coverage
  - Finds memory safety violations
KLEE: Performance on MSP430

- Why MSP430?
  - Popular, widely deployed
  - Security applications
  - Has clang support
- KLEE ported to 16-bit
- Evaluated 99 programs
  - 12 TI Community
  - 78 Github
  - 8 USB protocol stack
  - 1 Synthetic (cardreader)
- Average instruction coverage for MSP430 < 6%
  - Most programs < 1%
Challenges of MSP430 Code

- Peripheral access with I/O Ports
- Environment interaction via implicit memory mapping

```c
while (true) {
    if (*0x20)
        len = *0x20;
    BIS_SR(GIE);
    if (!*0x20)
        strncpy(dst, src, len);
}
PORT_2_ISR
PORT_2DIR = 0x0;
```

**Challenge #1**
Architecture Diversity

> 400 variants of MSP430

**Challenge #2**
Peripheral semantics
FiE on Firmware

- Challenge #1: Architecture Diversity
  - Handles over 400 variants of the MSP430

- Challenge #2: Peripheral semantics
  - Memory safety (21)
  - Peripheral misuse (1)

- Challenge #3: Interrupt-driven programs
- Verification
- Customizable

Firmware Source Code

Clang (MSP430)

LLVM Bitcode

Optimized Symbolic Execution Engine

Chip Layout Spec

Memory Spec

Interrupt Spec

Error Trace

verification
While adversary controls peripherals, allow users to supply custom libraries.

```c
while (true) {
    if (*0x20) {
        len = *0x20;
        __BIS_SR(GIE);
        if (!*0x20)
            strncpy(dst, src, len);
    }
}
```

**Chip Layout Table**

| addr | P1IN 0x20 1 |

**Memory Library**

```c
P1IN_READ:
fresh_symbolic()
```

**PORT_2_ISR**

```c
*0x22 = 0x0;
```
Challenges and Opportunities

- Verification
  - Outside scope of traditional symbolic execution
    - State space intractable
- Key Insight
  - Firmware state space much smaller
FiE on Verification

Infinite program paths
Analysis stuck executing already-seen states
Prevents verification

```c
while (true) {
    if (*0x20)
        len = *0x20;
    _BIS_SR(GIE);
    if (!*0x20)
        strncpy(dst, src, len);
}

PORT_2_ISR
*0x22 = 0x0;
```
FiE on Verification

- Log all execution states
- Pruning
  - Detect redundant states and terminate them
  - Redundant states; redundant successors
- Smudging
  - Replace frequently-changing concrete memory with symbolic
  - Complete
    - May have FPs

More details in the paper
FiE on Firmware

Challenge #1
Architecture Diversity

Challenge #2
Peripheral semantics

Challenge #3
Interrupt-driven programs

Chip Layout Spec
Memory Spec
Interrupt Spec

Optimized Symbolic Execution Engine

verification
Evaluation

- Amazon EC2
  - Automated tests (scripts available)
  - 50 minute runs
- Test Versions:
  - 16-bit KLEE
    - baseline
  - FiE
    - Symbolic + plugin
  - FiE + pruning
  - FiE + pruning + smudging

Corpus:
12 TI Community
1 Synthetic (cardreader)
8 USB protocol stack
78 Github
Bugfinding Results

- 22 bugs across the corpus (smudge)
  - Verified manually
  - 21 found in the MSP430 USB protocol stack
  - 1 misuse of flash memory
- Emailed developers
## Coverage Results

<table>
<thead>
<tr>
<th>Mode</th>
<th>Average % Coverage</th>
<th>False Positives</th>
<th>Verified</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>5.9</td>
<td>92</td>
<td>0</td>
</tr>
<tr>
<td>Symbolic</td>
<td>71.1</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>Prune</td>
<td>74.4</td>
<td>0</td>
<td>35</td>
</tr>
<tr>
<td>Smudge</td>
<td>79.4</td>
<td>1</td>
<td>53</td>
</tr>
</tbody>
</table>
Summary

Initiated work for MSP430 automated bugfinding

Modular, conservative symbolic execution

Supported verification and bugfinding

Download FiE

www.cs.wisc.edu/~davidson/fie