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

[Encrypted card data]

[Frisby et al., 2012]
Example Attack: WOOT 2012

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

Buffer Overflow!

[Frisby et al., 2012]
Embedded Systems: Lots of Attacks

**Design How-To**

**Embedded systems next for hack attacks**

Peter Clarke
2/26/2013 02:30 PM EST

**DILLON BERESFORD**

Exploiting Siemens Simatic S7 PLCs

During this presentation we will cover newly discovered Siemens Simatic S7-1200 PLC vulnerabilities. I plan to demonstrate how an attacker could impersonate the Siemens Step 7 PLC communication protocol using some PROFINET-FU over ISO-TSAP and take control.

---

**A Heart Device Is Found Vulnerable to Hacker Attacks**

By BARNABY J. FEDER
Published: March 12, 2008

**Kelly Jackson Higgins** December 27, 2011

1. Remotely starting a car via text message.

---

**PINPAPDWN**

Presented By:
Nils
Rafael Dominguez Vega

July 25

... Little Work on Detecting Vulnerabilities
Source code analysis is helpful on desktop

Could be transitioned to firmware

Embedded Systems: Lots of Attacks

... 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
• Represents program input as sets of constraints
• Explores multiple feasible paths for bugs
• Provides 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

```c
while (true) {
    if (P1IN)
        len = P1IN;
_BIS_SR(GIE);
    if (!P1IN)
        strncpy(dst, src, len);
}

PORT_2_ISR
P1DIR = 0x0;
```
Challenges of MSP430 Code

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

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

PORT_2_ISR
*0x22 = 0x0;

Challenge #1
Architecture Diversity

> 400 variants of MSP430
Challenges of MSP430 Code

value?

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

PORT_2_ISR
*0x22 = 0x0;

Challenge #1
Architecture Diversity

Challenge #2
Peripheral semantics
Challenges of MSP430 Code

Challenge #1
Architecture Diversity

Challenge #2
Peripheral semantics

Challenge #3
Interrupt-driven programs

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

PORT_2_ISR
*0x22 = 0x0;
FiE on Firmware

- Handles over 400 variants of the MSP430
- Bugfinding
  - Memory safety (21)
  - Peripheral misuse (1)
- Verification (53/99)
- Customizable
FiE on Firmware

**Firmware Source Code**
- Clang (MSP430)
- LLVM Bitcode

**Optimized Symbolic Execution Engine**
- Error Trace

**Challenge #1**
- Architecture Diversity

**Challenge #2**
- Peripheral semantics

**Challenge #3**
- Interrupt-driven programs

**Verification**
Variations in layout and capabilities of the microcontroller

- Memory size
- Memory region types
- Available interrupts
FiE on Architecture Diversity

Variations in layout and capabilities of the microcontroller

**Chip layout spec**

- **Memory size**
  - layout 0x10000
  - range 0x1080 0x10bf flash
  - range 0x10c0 0x10ff flash
  - addr P1IN 0x20 1
- **Available interrupts**
  - interrupt PORT2_ISR check_PORT2

Domain-specific specification language
Flat text file for manual manipulation
Script support for msp430-gcc export
while (true) {
    if (*0x20)
        len = *0x20;
    _BIS_SR(GIE);
    if (!*0x20)
        strncpy(dst, src, len);
}

PORT_2_ISR
*0x22 = 0x0;
Assume adversary controls peripherals
Allow users to supply custom libraries

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

PORT_2_ISR
*0x22 = 0x0;

Chip Layout Spec
addr P1IN 0x20 1

Memory Library
P1IN_READ:
  fresh_symbolic()
FiE on Interrupts

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

PORT_2_ISR
*0x22 = 0x0;

Chip Layout Spec
interrupt PORT2_ISR check_PORT2

Interrupt Library
Check_PORT2:
Interrupts On?
Port 2 Priority?

Adversary controls interrupts
Split state at every valid point
Challenges and Opportunities

Firmware Source Code

Clang (MSP430)

LLVM Bitcode

Optimized Symbolic Execution Engine

Error Trace

Challenge #1
Architecture Diversity

Challenge #2
Peripheral semantics

Challenge #3
Interrupt-driven programs

Chip Layout Spec

Memory Spec

Interrupt Spec

verification
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
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
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>
High-Challenge Programs

- FiE does well for small (but still useful!) programs
- For large programs, verification out of reach
- Reduce interrupts fired
  - **Conservative**: interrupts at each instruction
  - **Relaxed**: interrupts at each basic block

![Instruction Coverage (%)](chart)

**13 largest programs**
Future Work

• FiE breaks new ground
  – Not the final word by far

• One point in analysis design space
  – Dynamic testing
  – Concolic execution
  – Static analysis

• Language Design
Summary

Initiated work for MSP430 automated bugfinding

Modular, conservative symbolic execution

Supported verification and bugfinding

Download FiE

www.cs.wisc.edu/~davidson/fie
• While Pruning:
  – Check unique values for each memory object
  – If above threshold, replace with wildcard (*)

• Makes pruning easier
  – Redundant states sooner

• Complete
  – May cause false positives

Q: Smudging example

\[
\text{L_{loop}}: i = 0
\]

\[
\text{L_{loop}}: i = 1
\]

\[
\text{L_{loop}}: i = 2
\]

\[
\ldots
\]

\[
\text{L_{loop}}: i = *
\]

\[
\text{L_{end}}: i = *
\]

\[
\text{L_{loop}}: i = *
\]
Q: Corpus Code Size

- < 201 LOC
- > 200 LOC
Q: Why didn’t you find more bugs?

• It’s easy to get hobbyist code

• The production code that we do have indicates a problem

• The tractability of hobbyist code indicates an opportunity for deeper analysis
Q: What about Coverity?

- Commercial analysis tool
- Static Analysis
- Has an MSP430 target
  - License forbids published comparison
Q: What Does this Mean for KLEE?

- KLEE is a great tool
  - The performance is great
  - The code is great
- We use it in a way that it wasn’t intended for