FiE on Firmware

Finding Vulnerabilities in Embedded Systems using Symbolic Execution

Drew Davidson
Ben Moench
Somesh Jha
Thomas Ristenpart

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

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

Command

Secret Key

Buffer Overflow!

[Frisby et al., 2012]

Embedded Systems: Lots of Attacks

designlines INTERNET OF THINGS

Design How-To

Embedded systems next for hack attacks

Peter Clarke

2/26/2013 02:30 PM EST

NO RATINGS
LOGIN TO RATE

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.

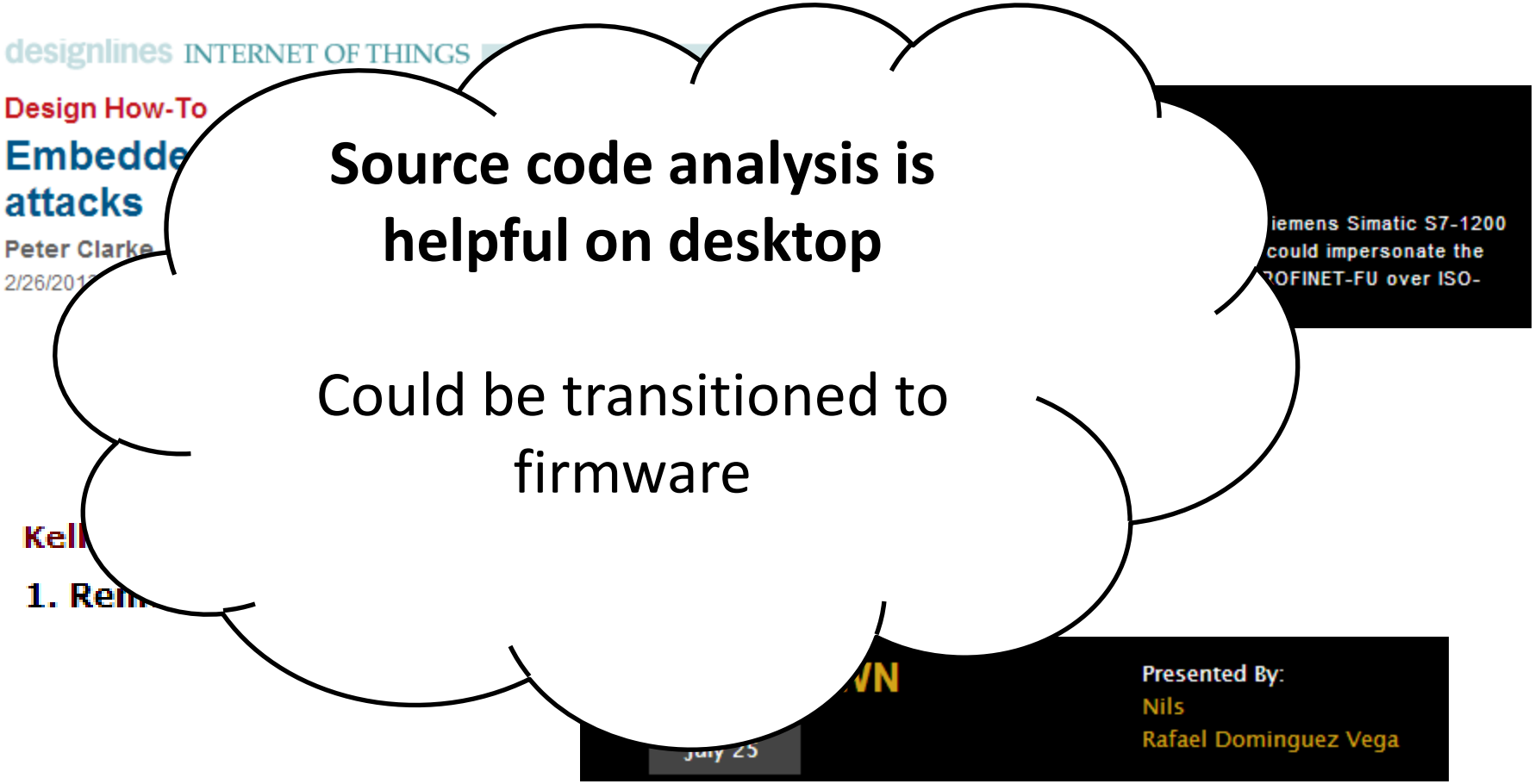
PINPADPWN

July 25

Presented By:
Nils
Rafael Dominguez Vega

… Little Work on Detecting Vulnerabilities

Embedded Systems: Lots of Attacks

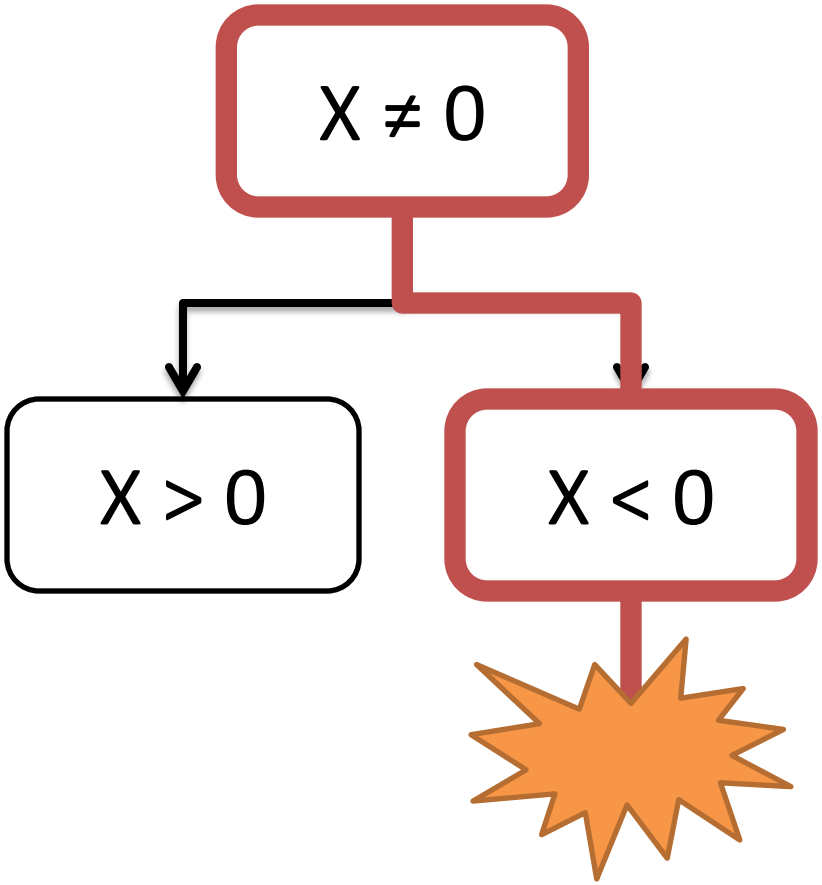


Source code analysis is helpful on desktop

Could be transitioned to firmware

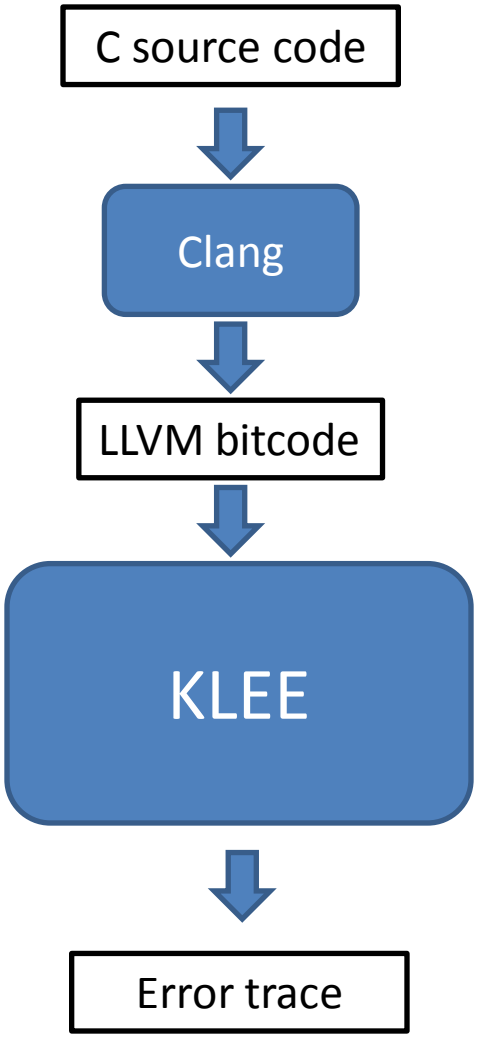
… Little Work on Detecting Vulnerabilities

Symbolic Execution



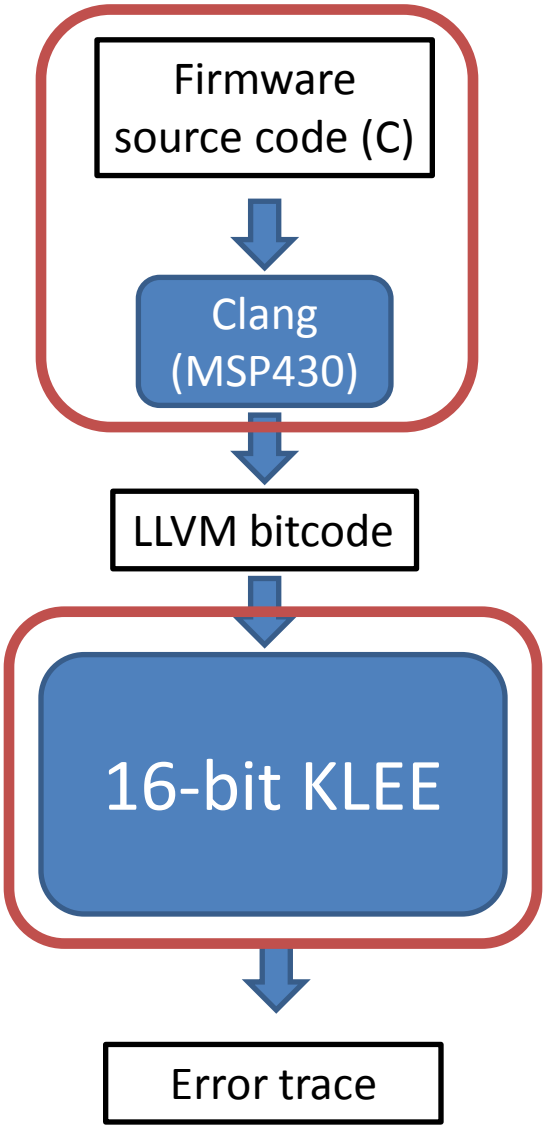
- Represents program input as sets of constraints
- Explores multiple feasible paths for bugs
- Provide detailed trace to vulnerability

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

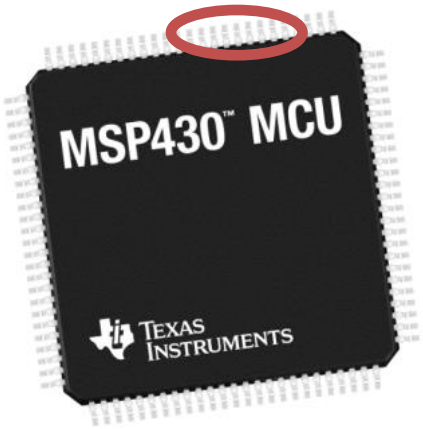
KLEE: Performance on MSP430



- 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**

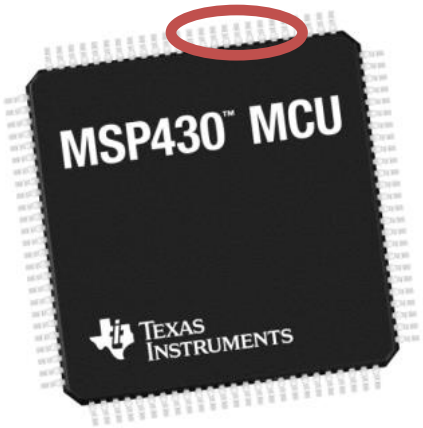



```
while (true){
   if ( P1IN)
      len =  P1IN;
   _BIS_SR(GIE);
   if (! P1IN)
      strncpy(dst,src,len);
}
```

PORT_2_ISR

```
P1DIR = 0x0;
```

Challenges of MSP430 Code



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

Challenge #1
Architecture Diversity

> 400 variants of MSP430

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

   PORT_2_ISR
*0x22 = 0x0;
```

Challenges of MSP430 Code

value?

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

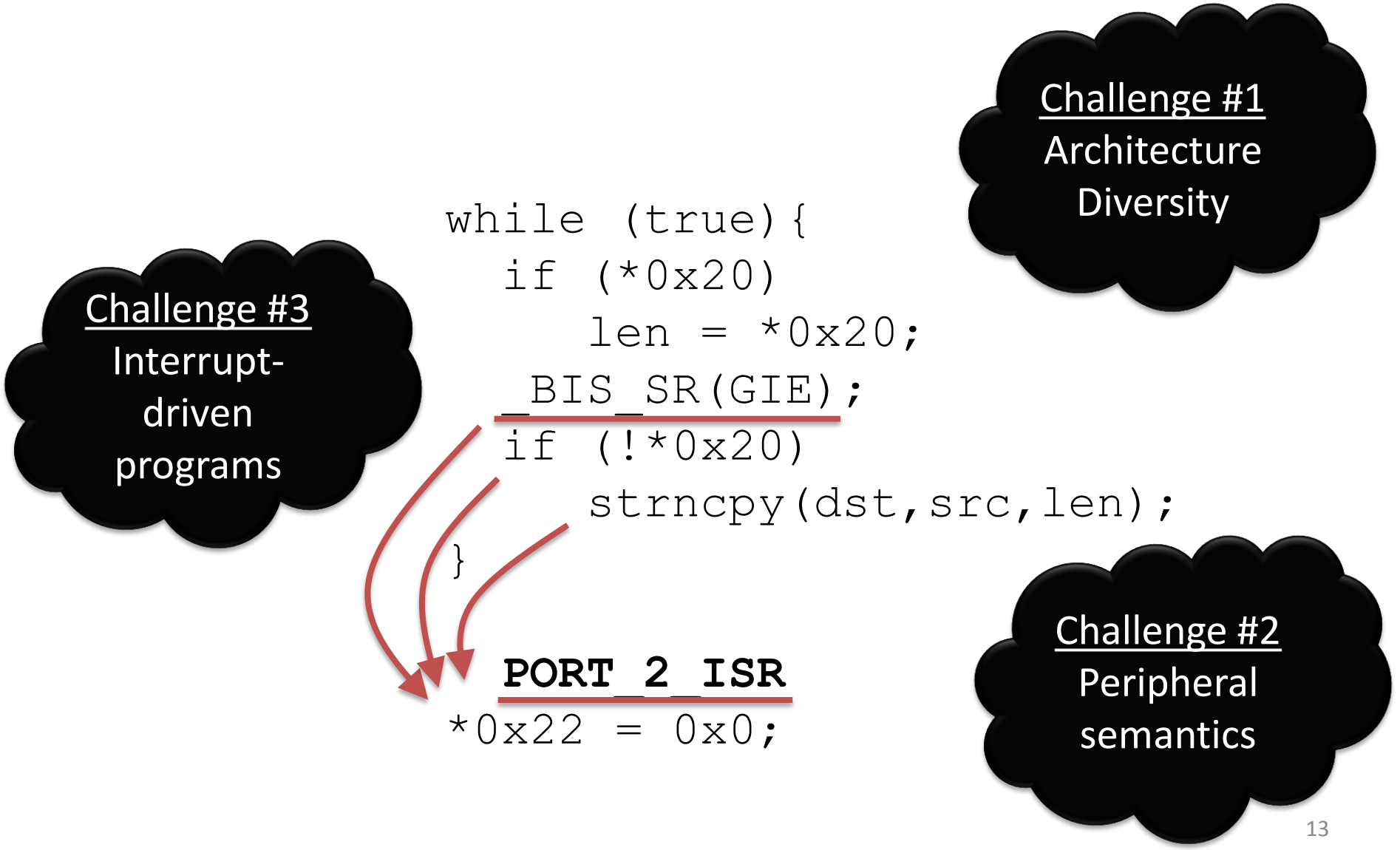
PORT_2_ISR

```
*0x22 = 0x0;
```

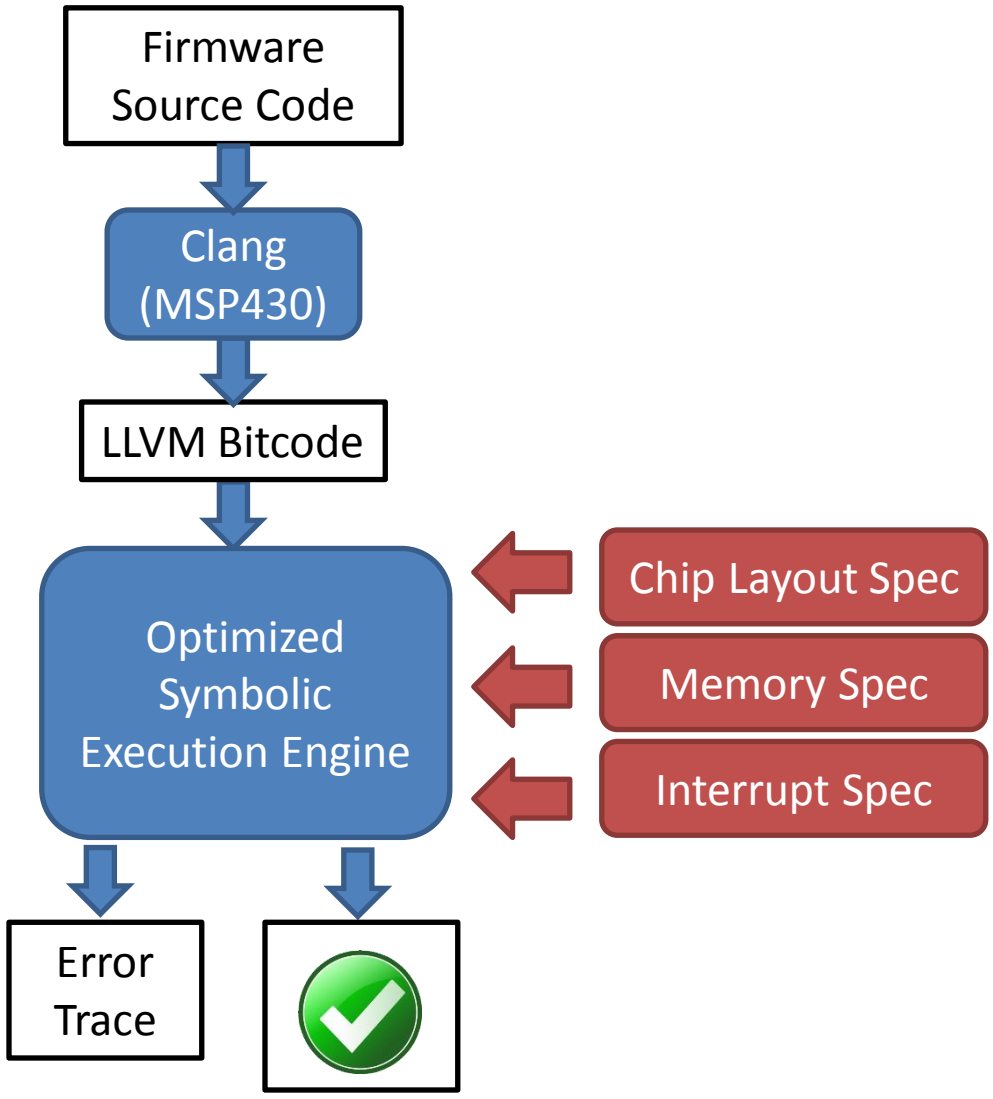
Challenge #1
Architecture
Diversity

Challenge #2
Peripheral
semantics

Challenges of MSP430 Code

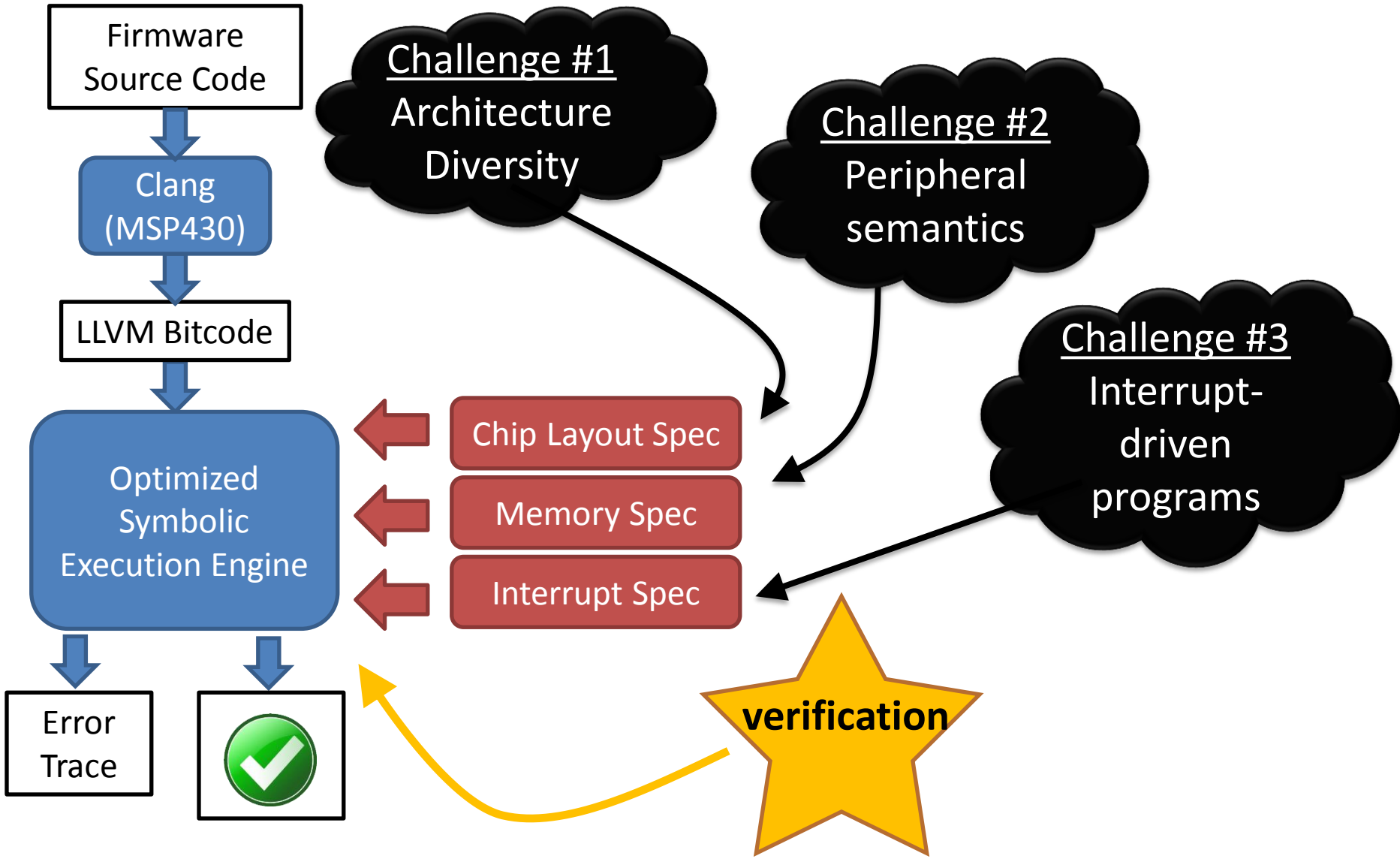


FiE on Firmware



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

FiE on Firmware



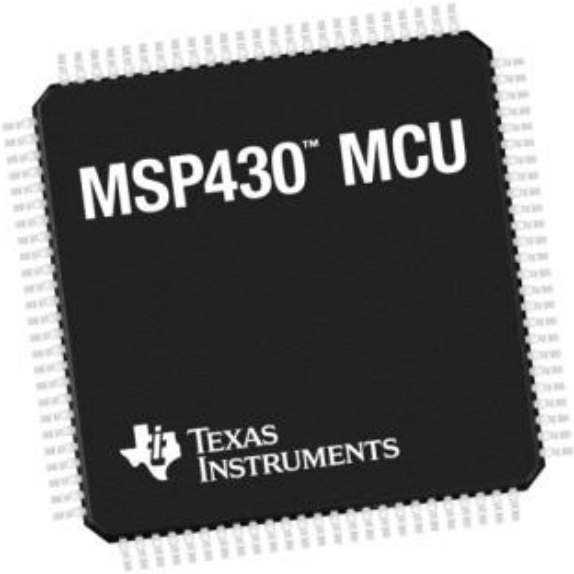
FiE on Architecture Diversity

Variations in layout and capabilities of the microcontroller

Memory size

Memory region types

Available interrupts



FiE on Architecture Diversity

Chip layout spec

Memory size

Memory region types

Available interrupts

Variations in layout and capabilities of the microcontroller

```
layout 0x10000
range 0x1080 0x10bf flash
range 0x10c0 0x10ff flash
addr P1IN 0x20 1
interrupt PORT2_ISR check_PORT2
```

Domain-specific specification language
Flat text file for manual manipulation
Script support for msp430-gcc export

FiE on Memory

Chip Layout Spec

```
addr P1IN 0x20 1
```

Memory Library

```
P1IN_READ:
      ???
```

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

   PORT_2_ISR
*0x22 = 0x0;
```

FiE on Memory

Assume adversary controls peripherals
Allow users to supply custom libraries

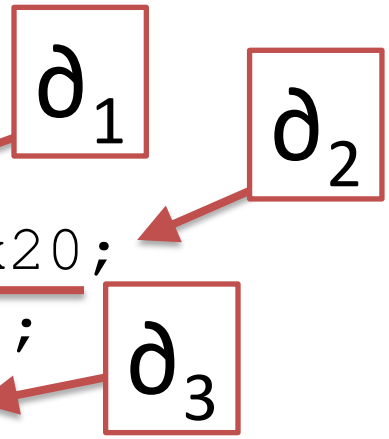
Chip Layout Spec

```
addr P1IN 0x20 1
```

Memory Library

```
P1IN_READ:
fresh_symbolic()
```

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

   PORT_2_ISR
*0x22 = 0x0;
```



FiE on Interrupts

Chip Layout Spec

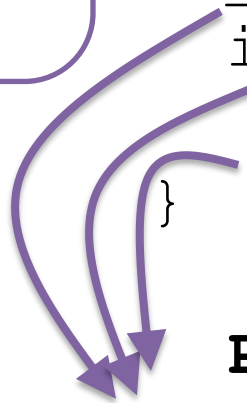
```
interrupt PORT2_ISR check_PORT2
```

Interrupt Library

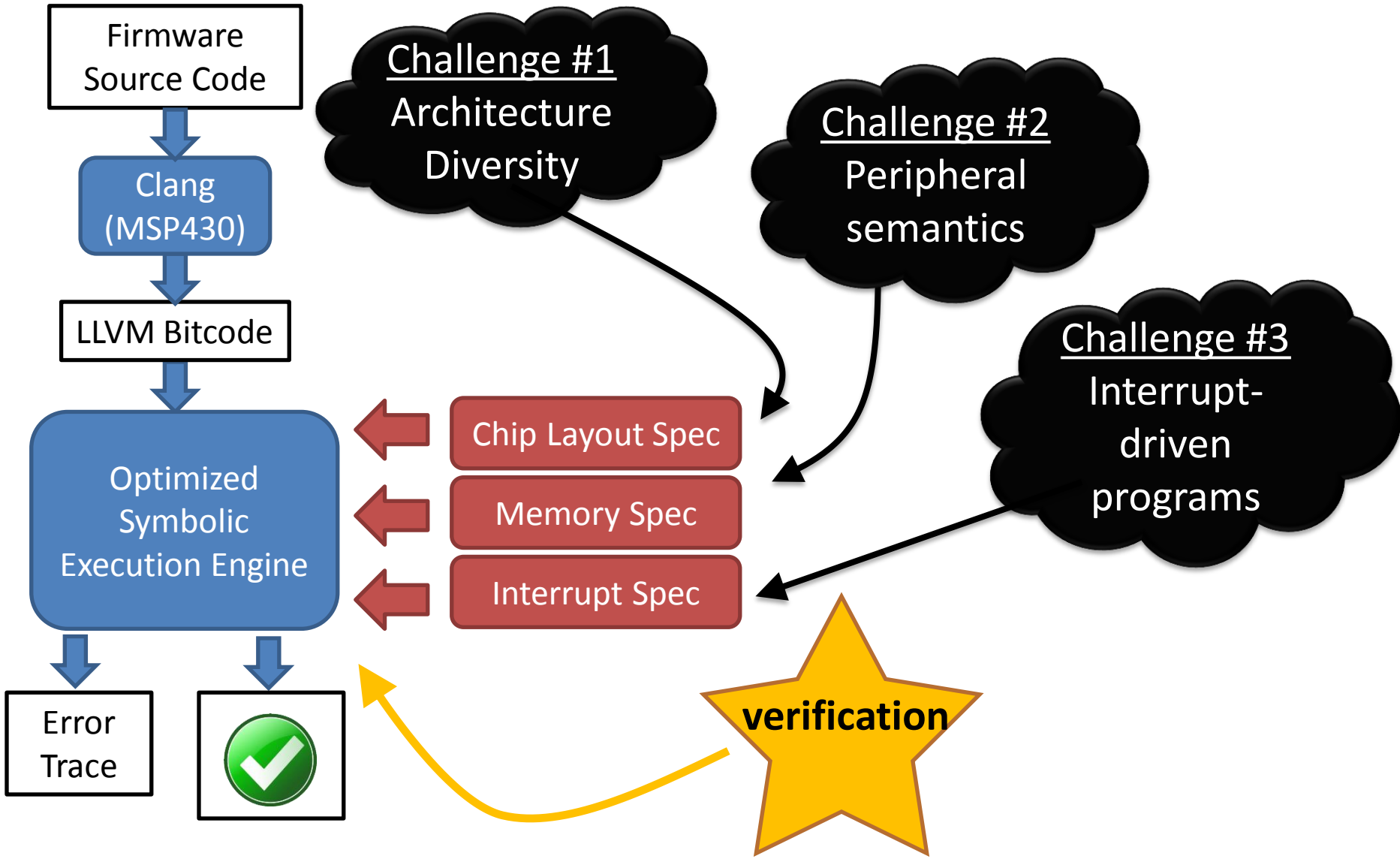
Check_PORT2:
Interrupts On?
Port 2 Priority?

Adversary controls interrupts
Split state at every valid point

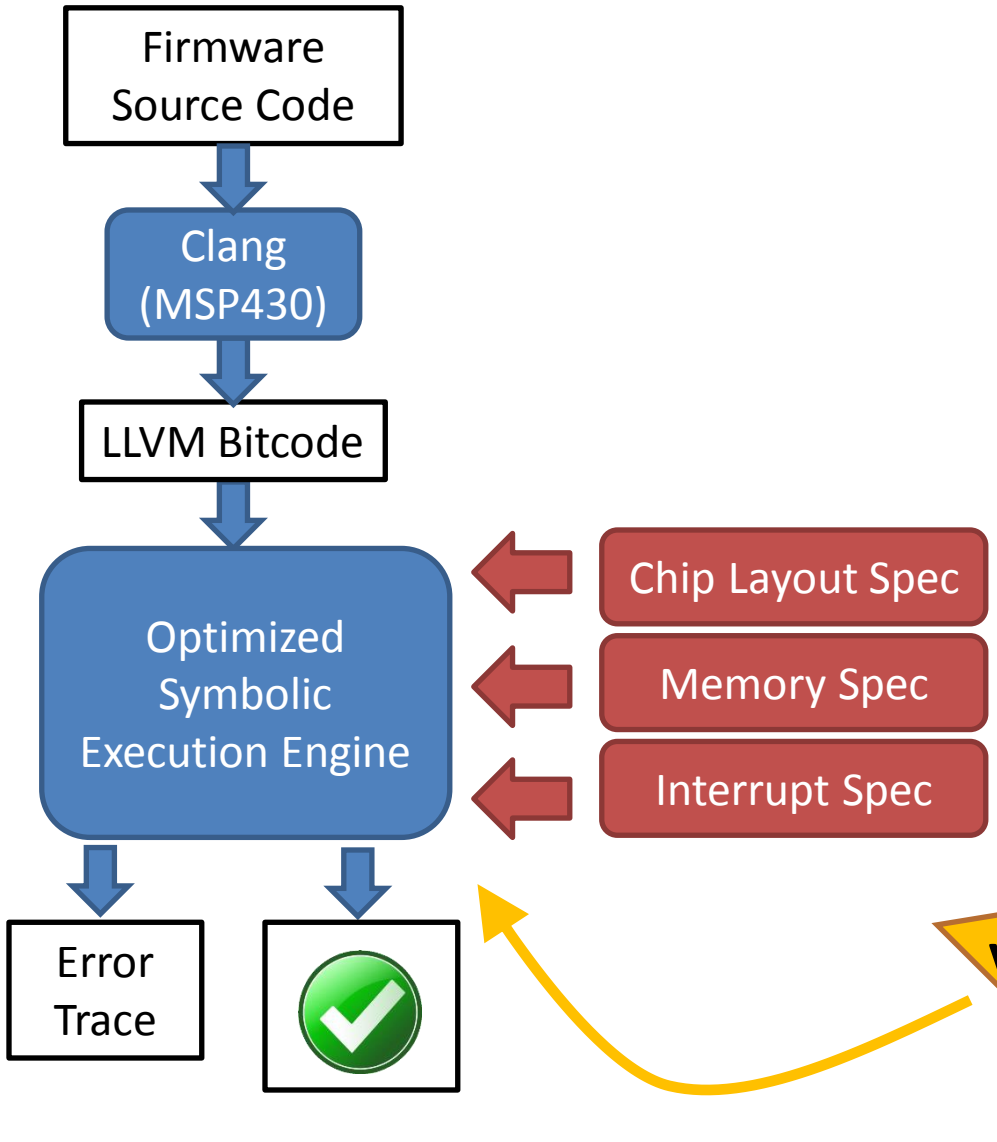
```
while (true){
  if (*0x20)
     len = *0x20;
  _BIS_SR(GIE);
  if (!*0x20)
     strncpy(dst,src,len);
}

  PORT_2_ISR
*0x22 = 0x0;
```



Challenges and Opportunities

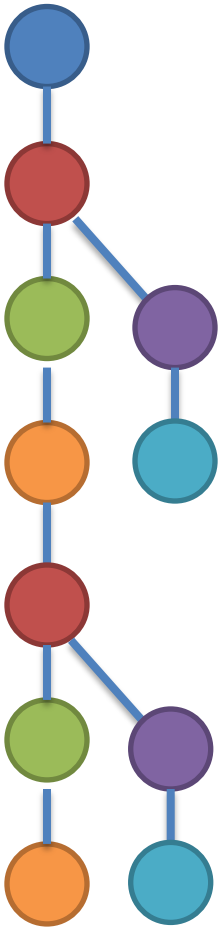


Challenges and Opportunities



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

FiE on Verification



Infinite program paths

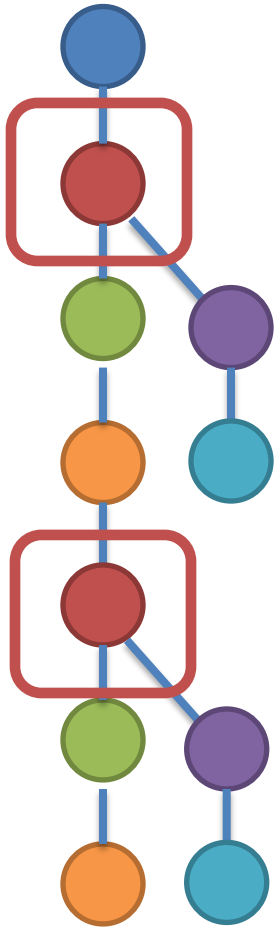
Analysis stuck executing already-seen states

Prevents verification

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

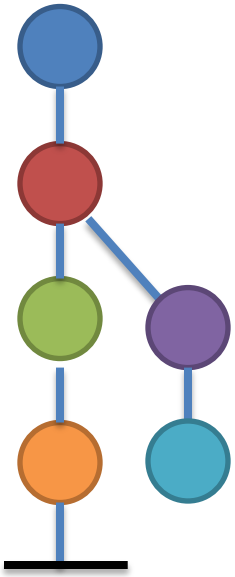
```
   PORT_2_ISR
*0x22 = 0x0;
```

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

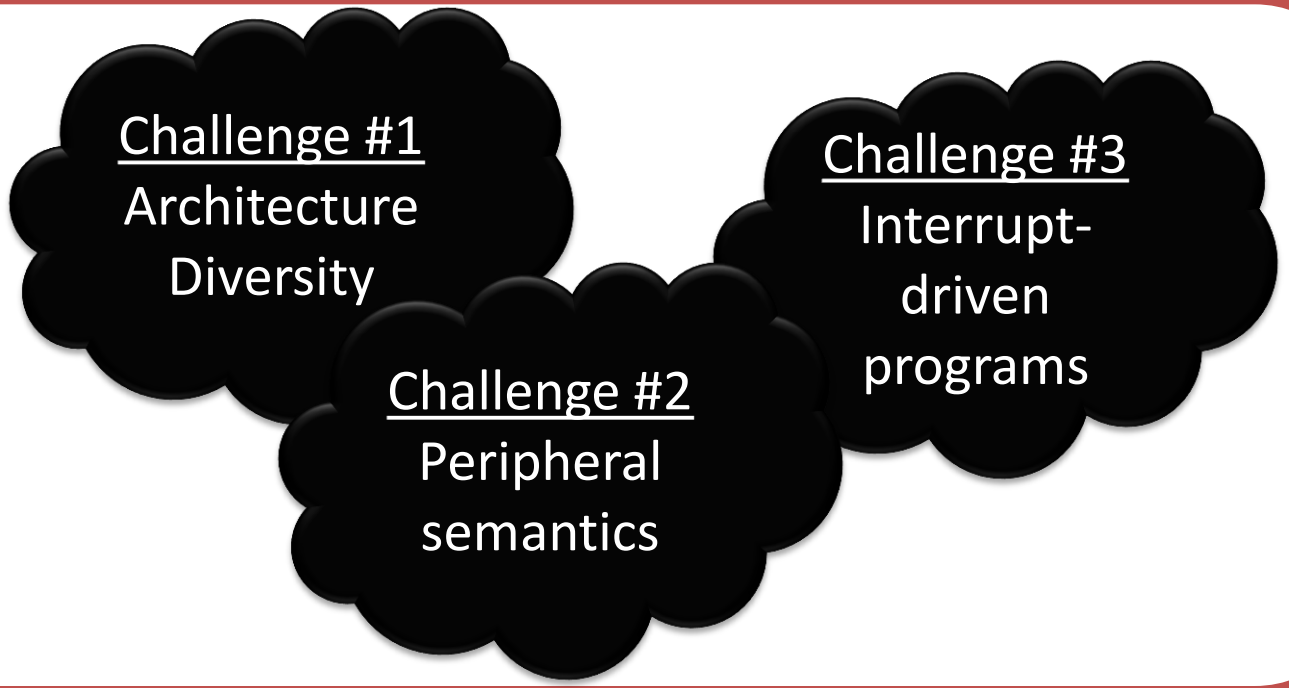
More details in the paper

FiE on Firmware

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

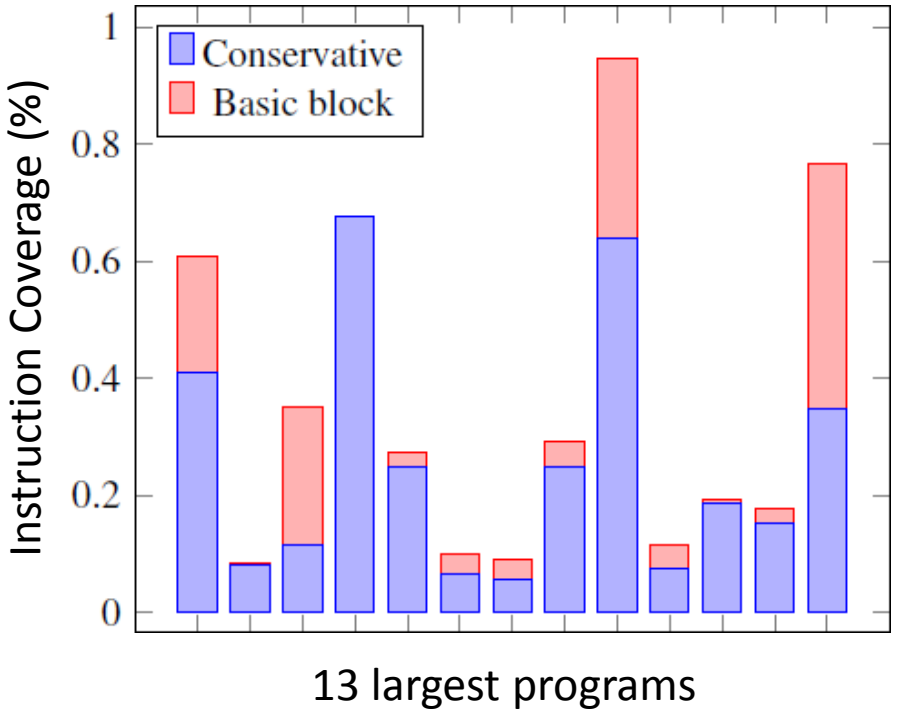
Bugfinding Results

- 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

Mode	Average % Coverage	False Positives	Verified
Baseline	5.9	92	0
Symbolic	71.1	0	7
Prune	74.4	0	35
Smudge	79.4	1	53

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

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

Thanks!

Summary

Initiated work for MSP430 automated bugfinding

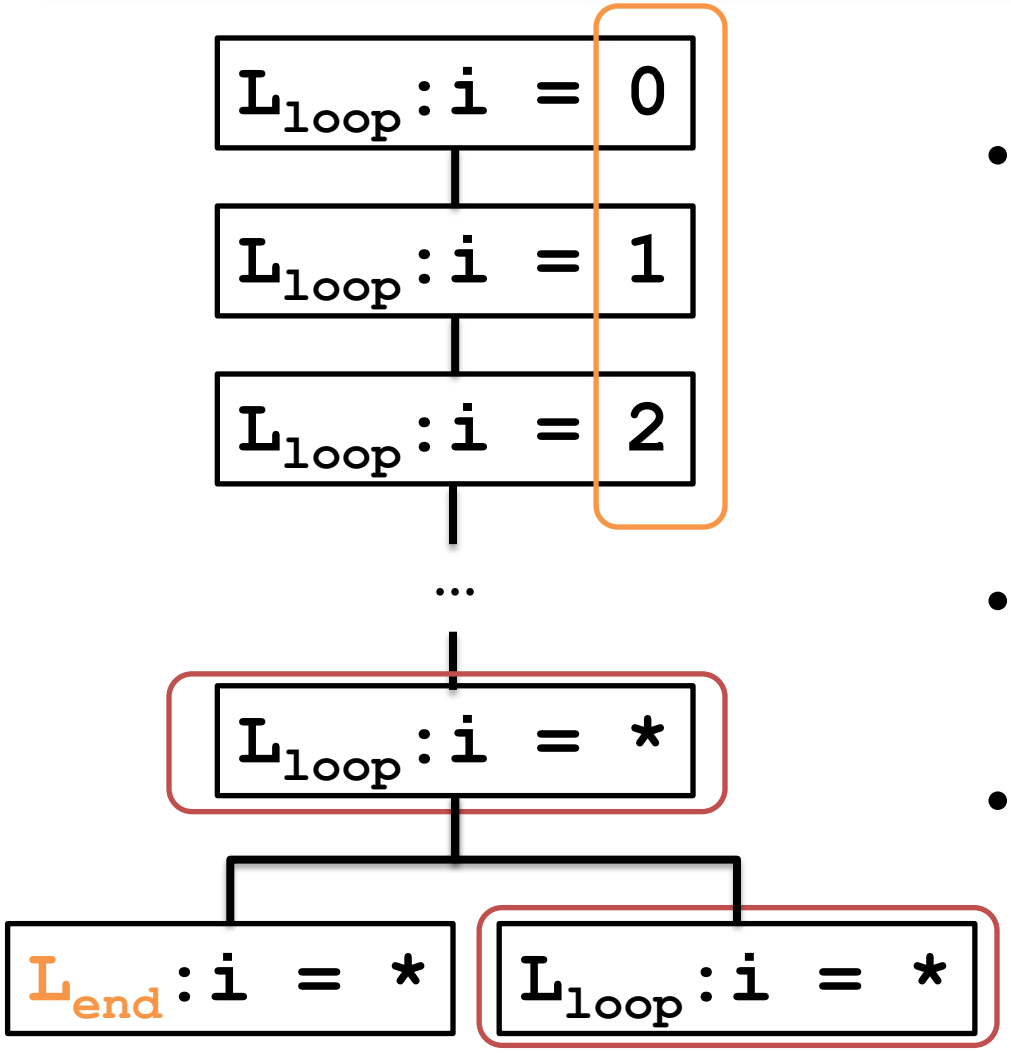
Modular, conservative symbolic execution

Supported verification and bugfinding

Download FiE

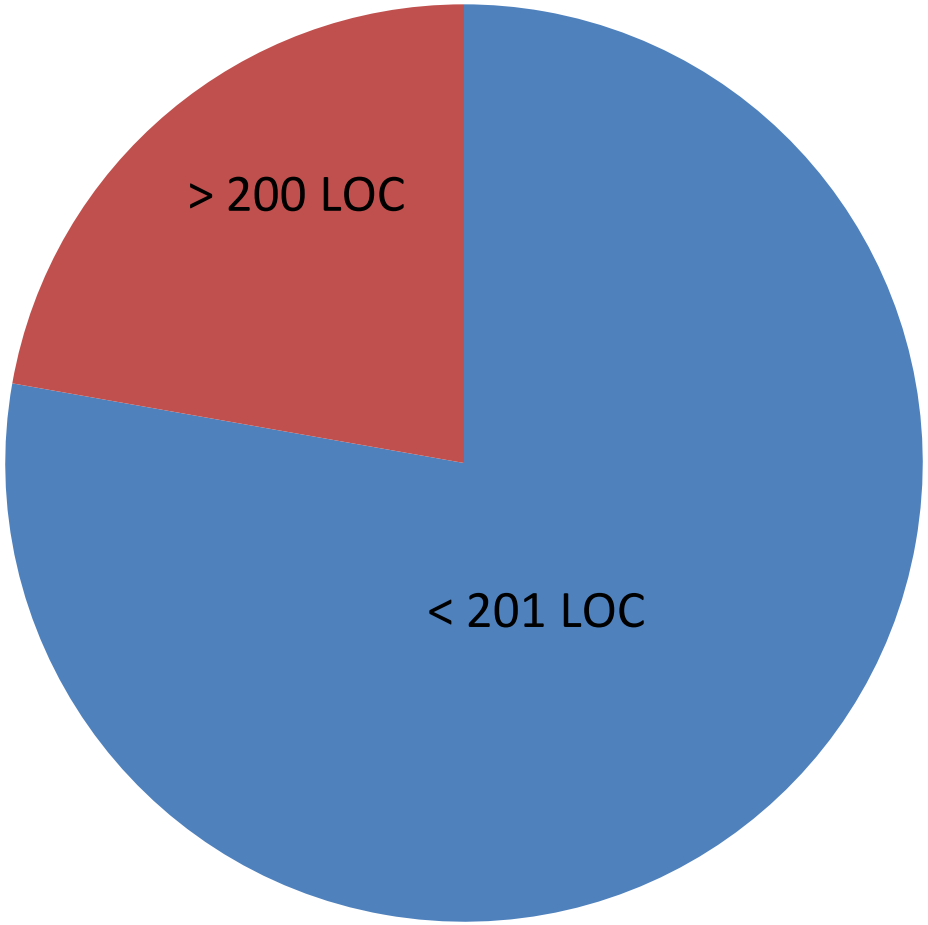
www.cs.wisc.edu/~davidson/fie

Q: Smudging example



- 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: Corpus Code Size



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