Automatic Firmware Emulation through Invalidity-guided Knowledge Inference --μEmu

Wei Zhou , Le Guan, Peng Liu and Yuqing Zhang

Bugs in MCU-based devices are being exploited to jeopardize our cyber-physical systems

Multiple Vulnerabilities in Treck TCP/IP Stac Arbitrary C

MS-ISAC ADVISORY NUMB
2020-171

DATE(S) ISSUED:
12/21/2020

OVERVIEW:

Multiple vulnerabilities have been c
code execution. Treck TCP/IP Stack
are widely used. Successful exploita
arbitrary code in the context of the
could install programs; view, chang
been configured to have fewer user
have less impact than if it was confi

THREAT INTELLIGENCE:

There are currently no reports of th

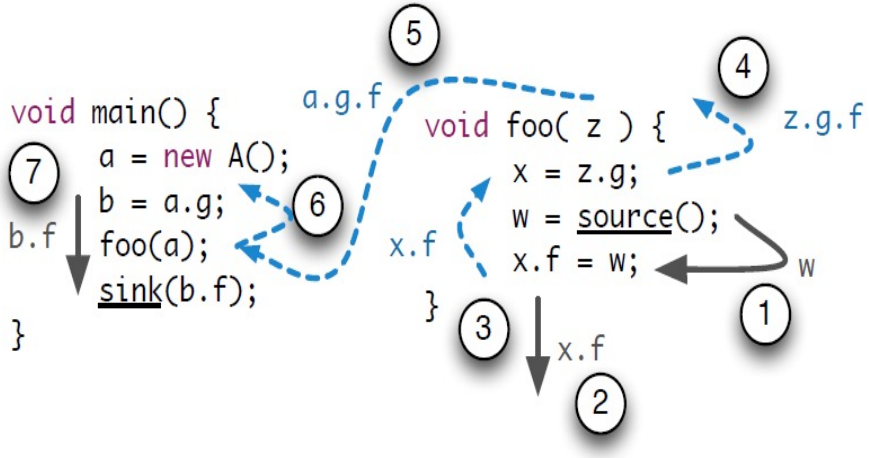
The hack attack led to failures in plant equipment and forced the fast shut down of a furnac

Hacking risk leads to recall of 500,000 pacemakers due to patient death fears

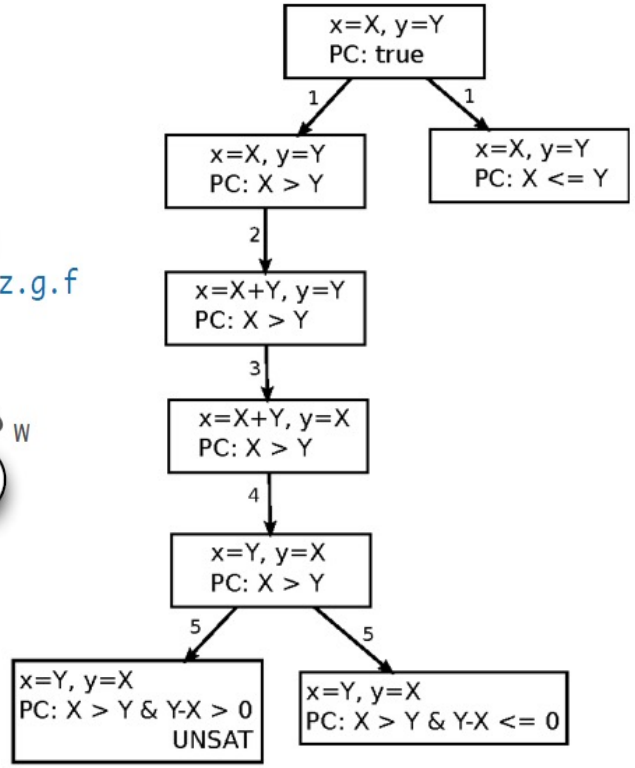
FDA overseeing crucial firmware update in US to patch security holes and prevent hijacking of pacemakers implanted in half a million people

University of Chinese Academy of Sciences

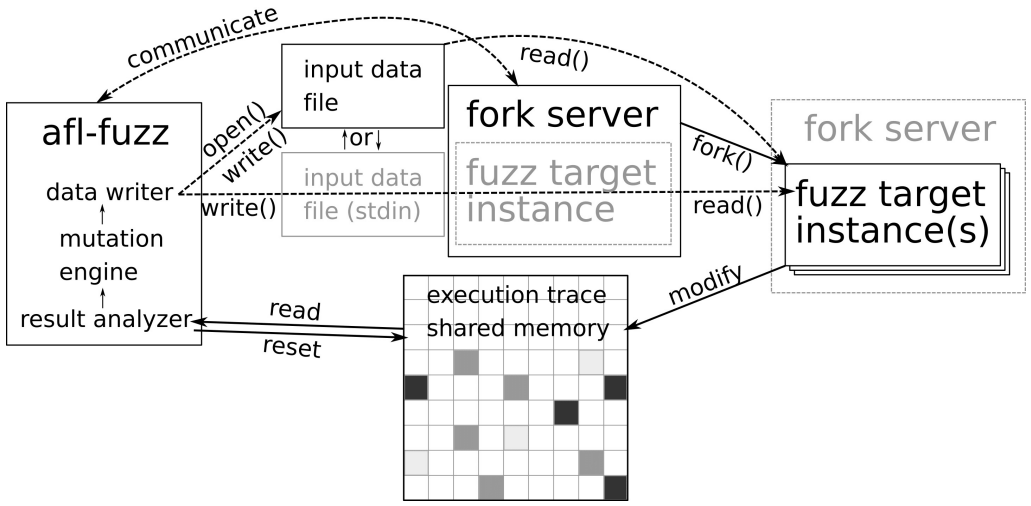
Dynamic analysis is popular in bug finding



Dynamic Taint Analysis

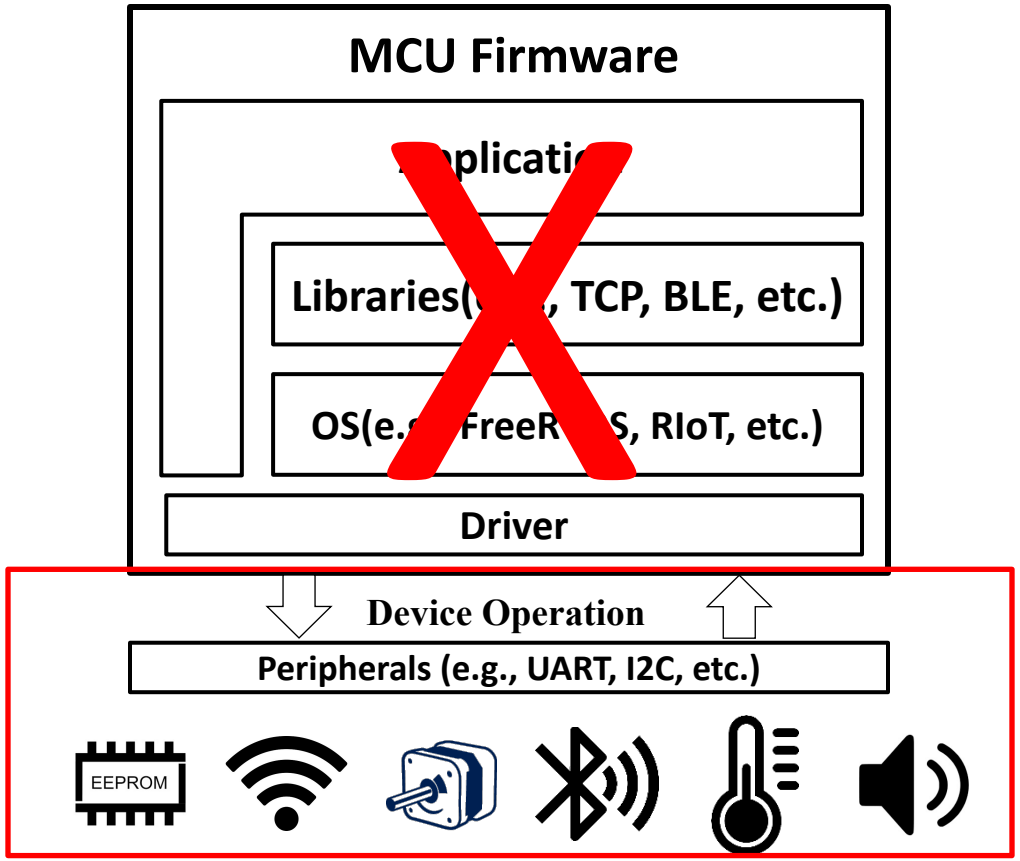


Dynamic Symbolic Execution



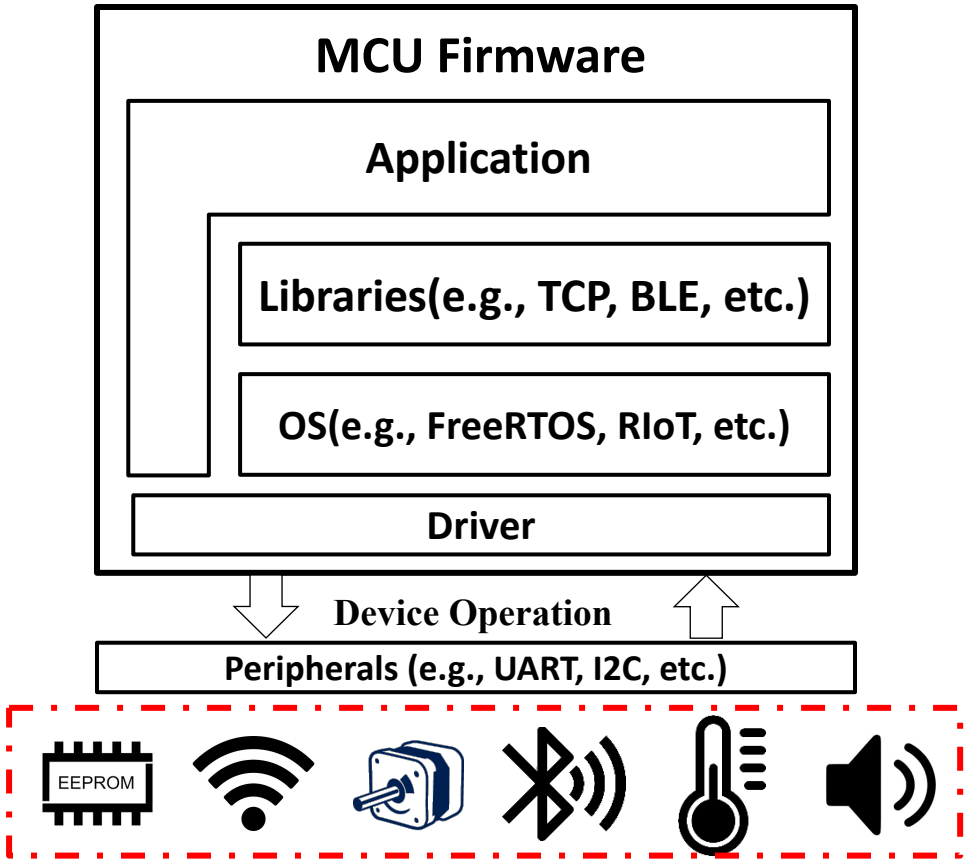
Fuzz Testing

Existing dynamic approaches fail to test MCU firmware



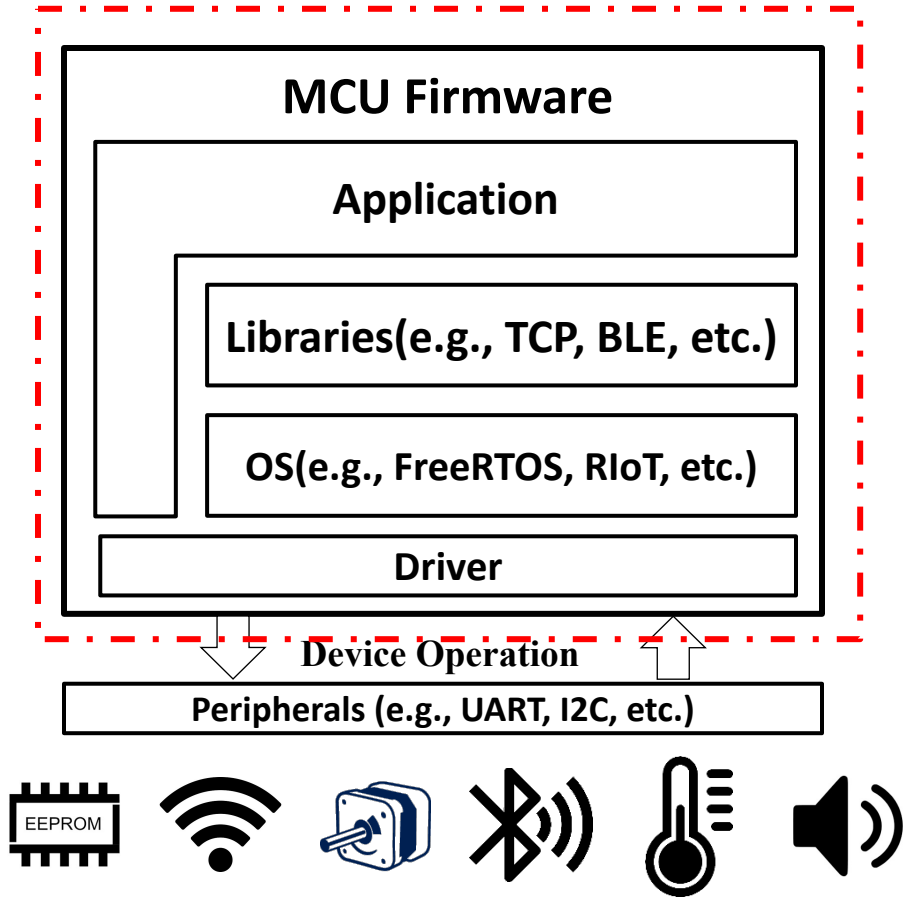
- Real hardware does not provide rich run-time information.
  - WYCNWYC [NDSS'18]
- Emulating firmware is hard
  - QEMU only supports 10+ MCU devices
  - Problem: peripheral diversity

Existing solution: manual peripheral modeling



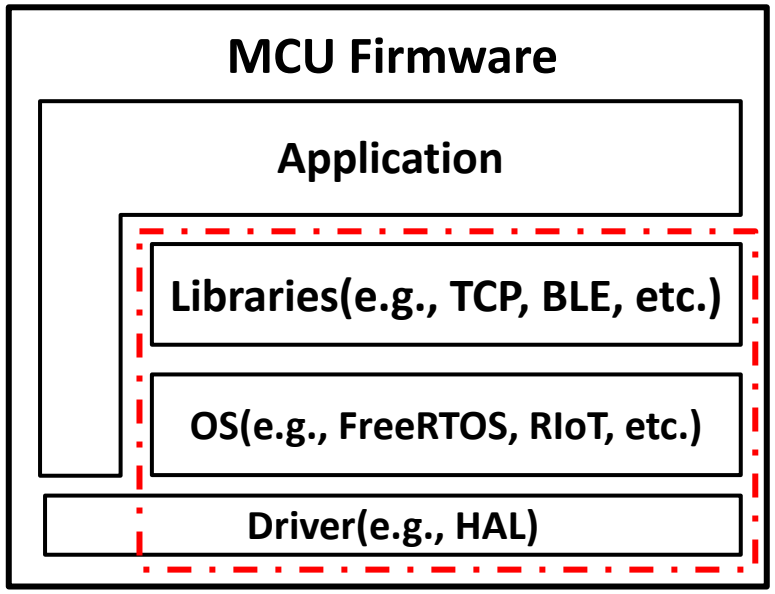
- Idea: develop peripherals models based on chip manuals like QEMU.
  - Can be useful dealing with a particular device model
- Disadvantages
  - Unscalable
  - Significant manual efforts

Existing solution: hardware-in-the-loop



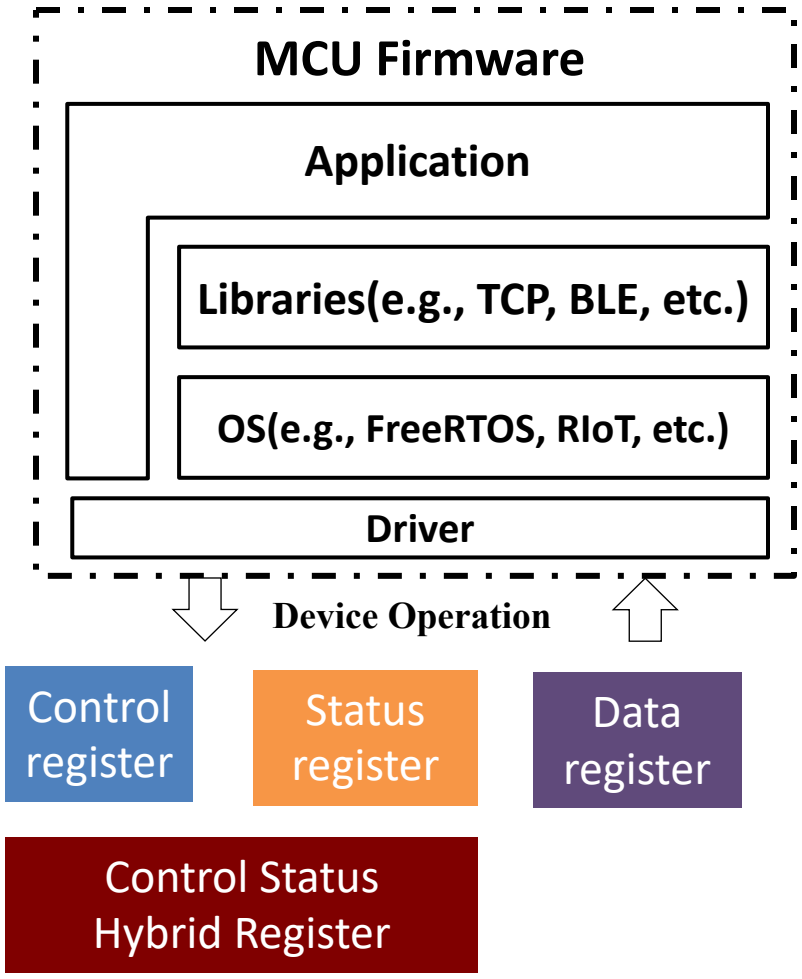
- Idea: forward the interactions with peripherals to the real hardware.
  - Avatar [NDSS'14]
- Disadvantages
  - Rely on real hardware
  - Slow
  - Unscalable

Existing solution: hardware abstraction



- Idea: replace the hardware abstraction layer (HAL) with software stubs that provide the same functions
  - HALucinator[SEC'20]
- Disadvantages
  - Unable to test peripheral-dependent code
  - HAL does not always available for all firmware

Existing solution: automatic peripheral modeling



- Idea: observe peripheral-firmware interactions and infer peripheral models
  - P2IM [SEC'20]
    - Categorize the peripheral registers based on access pattern
    - Respond to peripheral access requests based on register category and heuristics
- Disadvantages
  - Register mis-categorization
  - Blindly guess the response for some registers is low accuracy

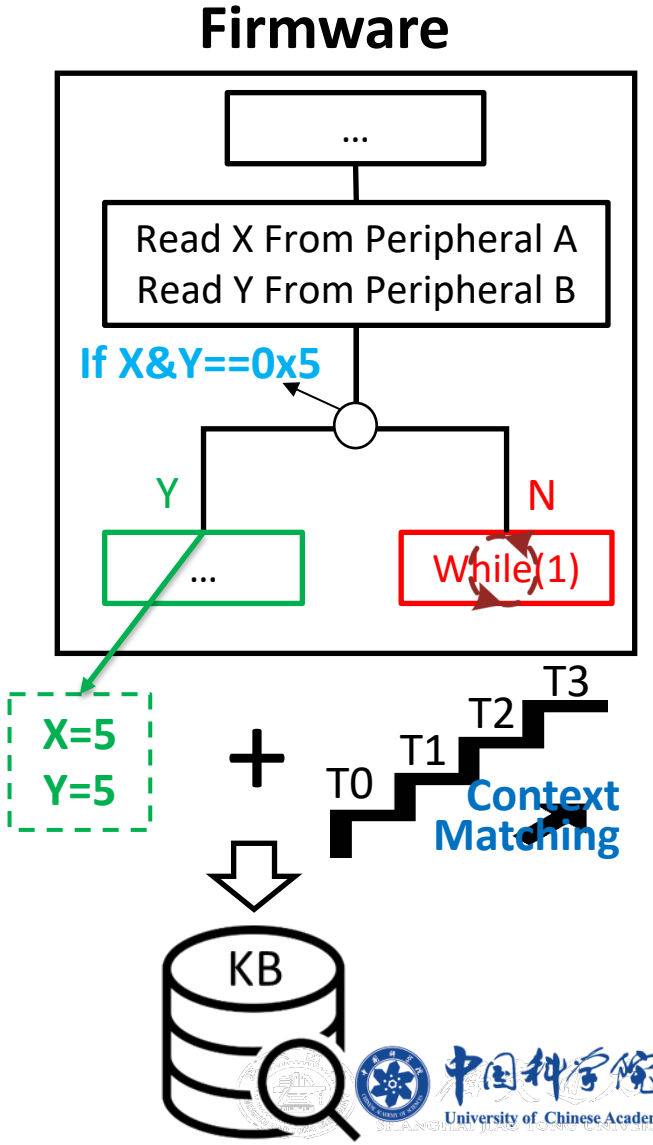
Existing solution: automatic peripheral modeling

- Idea: observe peripheral-firmware interactions and infer peripheral models
  - P2IM [
  - on access
  - ed on
  - categorization
  - Blindly guess the response for some registers is low accuracy

We propose a new automatic peripheral modeling approach with less heuristics and higher accuracy

Observation and high-level idea

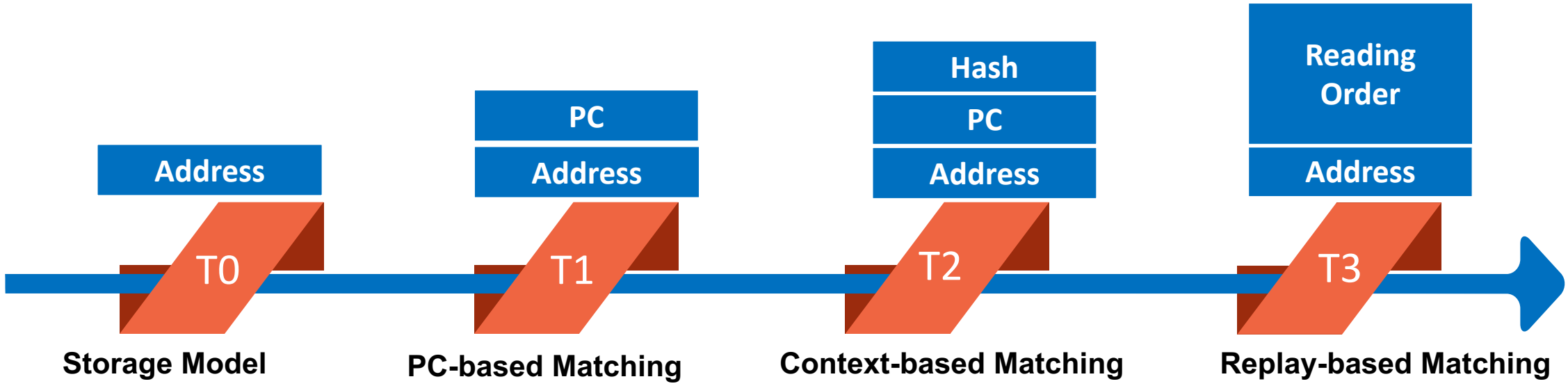
- Observation 1: If a response is incorrectly fed to the firmware, the error will eventually be reflected as an invalid execution state (e.g., a dead loop)
- Observation 2: A response can be reused when the same peripheral is access again under the same context.
- Idea
  - Represent responses to peripheral accesses as symbolic values and then use symbolic execution to explore the firmware.
  - Detect potential invalid paths
  - Build a knowledge base (KB) that guides the execution to avoid invalid paths
    - The knowledge base **is a tiered cache** system



Invalid States

- **Infinite Loop -** If the firmware execution encounters an unrecoverable error, it will halt itself by running **a simple infinite loop**.
- **Long Loop -** It is common that firmware waits for a certain value in a peripheral register, which is implemented **via a long loop** that pulls the value. A wrong response leads the execution to finish the long loop with an error code returned.
- **Invalid Memory Access -** Firmware uses response from peripheral to address memory. A wrong response lead to **unmapped memory access**.
- **User-defined Invalid Program Points -** µEmu is interactive. It allows analysts to specify program points to be avoided.

Context matching of the tiered cache system



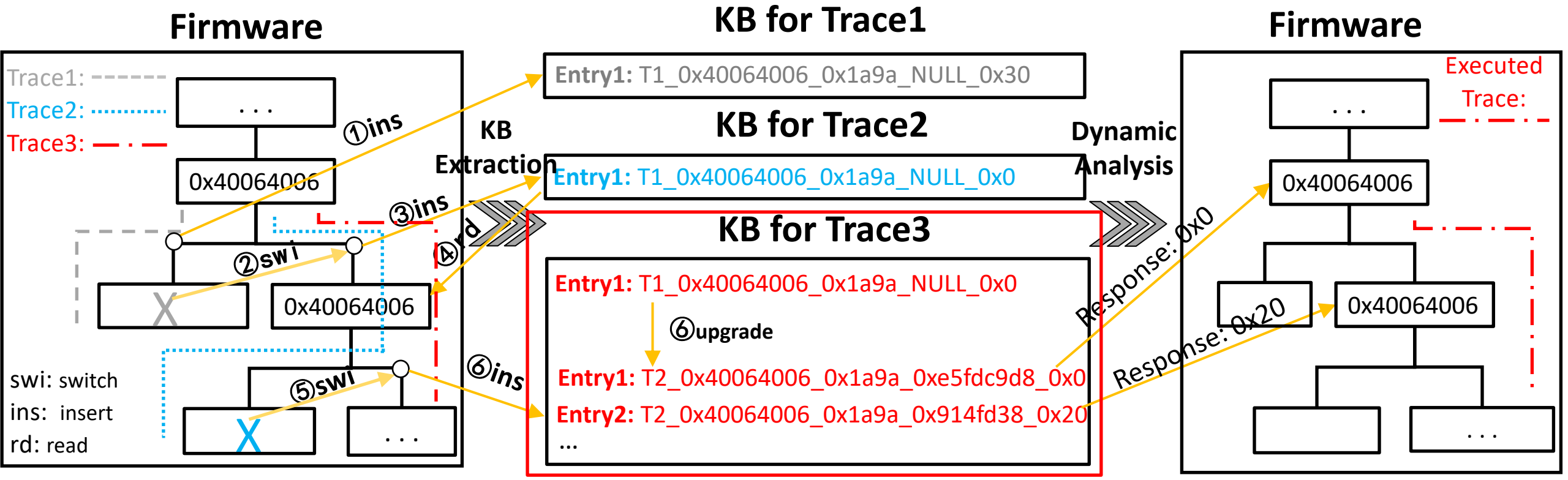
Entry	T0_addr_{written value}
Value	The most recent value written to it.

Entry	T1_addr_pc_value
Value	1.Peripheral register address 2.Reading PC

Entry	T2_addr_pc_Hash_value
Value	1.Peripheral register address 2.Reading PC **3.contextHash: call stack and function arguments.**

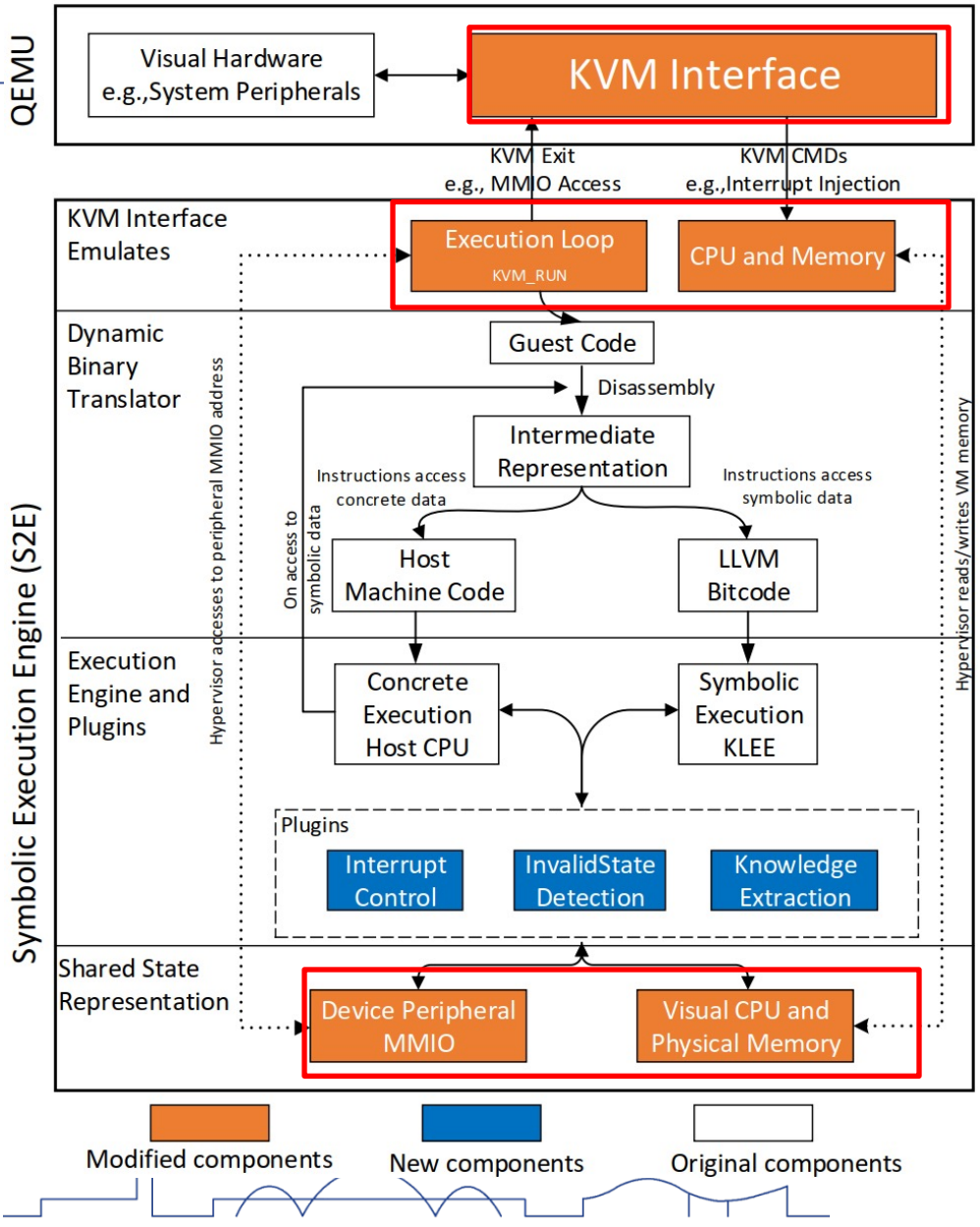
Entry	T3_addr_vec{v1,v2,..}
Value	1.Peripheral register address **2.Vector of value sorted by reading order**

A running example



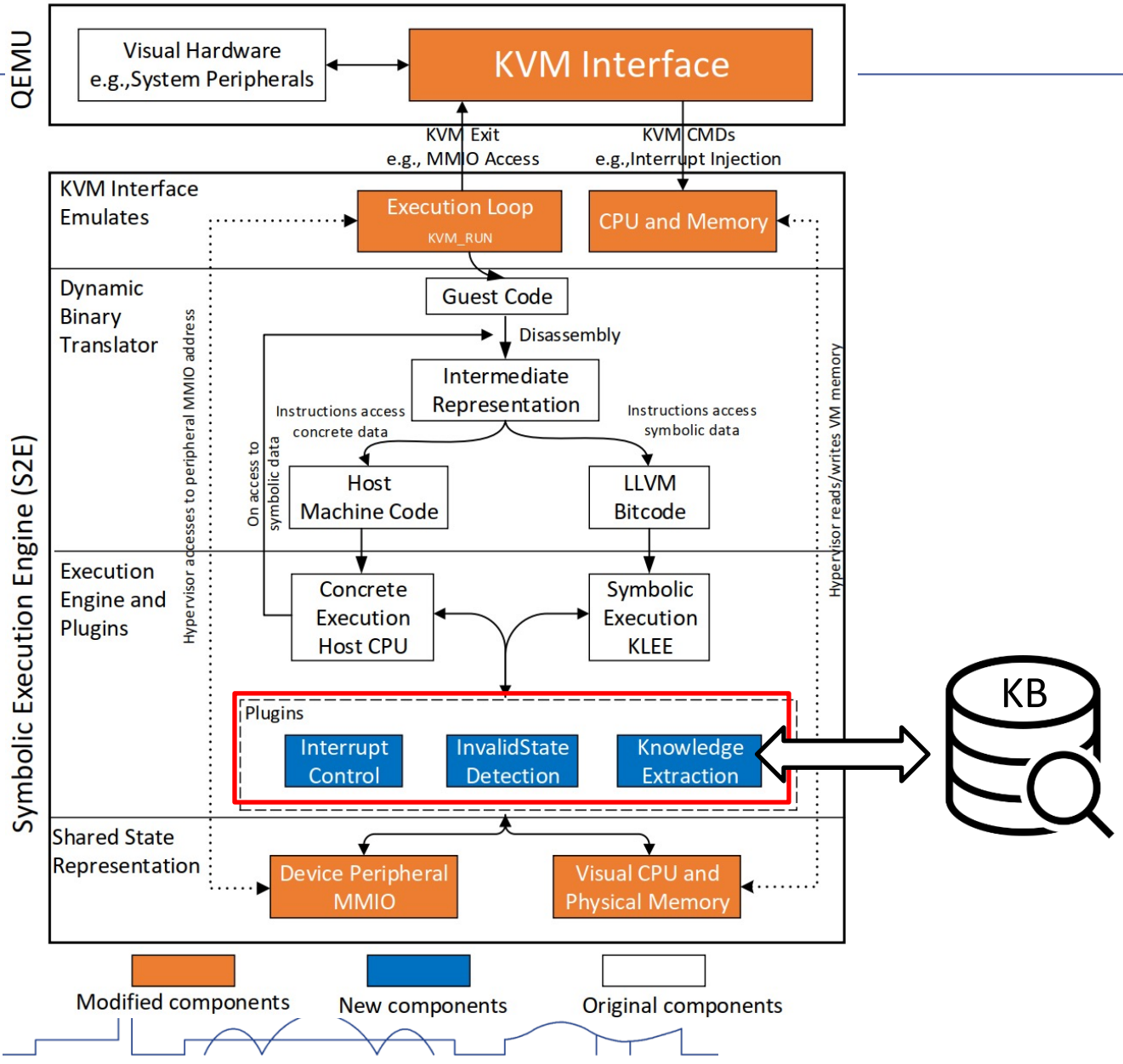
In the example, two branches both correspond to reading the peripheral register mapped at 0x40064006 at PC 0x1a9a.

Implementation - ARM support for S2E



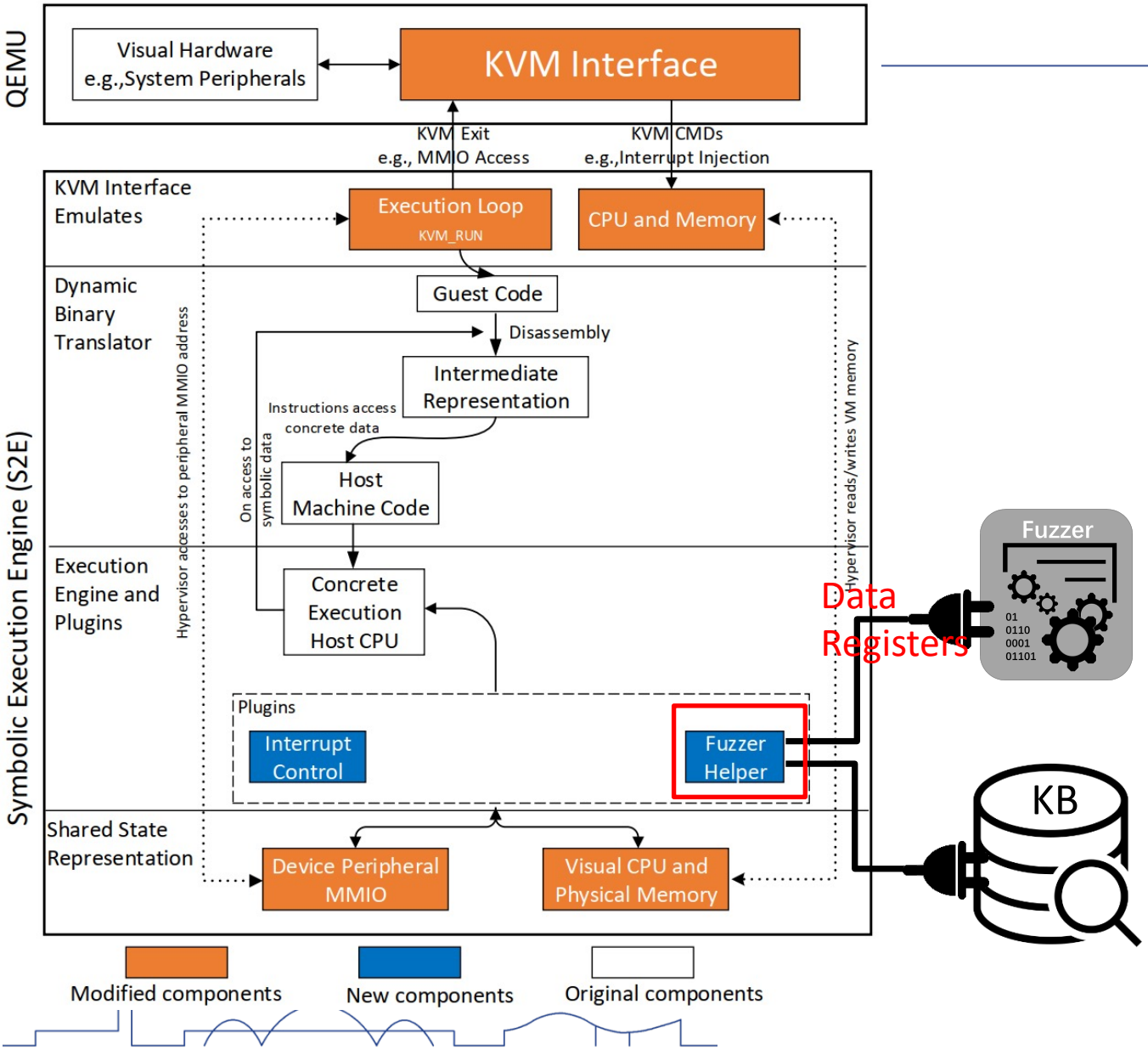
- S2E
  - Full system concolic execution engine based on QEMU and KLEE
  - Useful plugins and APIs
  - No ARM support
- ARM CPU Emulation
  - ARM TCG front-end
  - Low-level plugin hook
- KVM interface emulation
  - Register Initialization
  - Memory Registration
  - System Peripheral State Synchronization

Implementation - µEmu plugins



- Invalid States Detection
  - Invalid states notification
  - Path Switch
- Knowledge Extraction
  - Build and use KB
- Interrupt Control
  - Trigger external IRQs

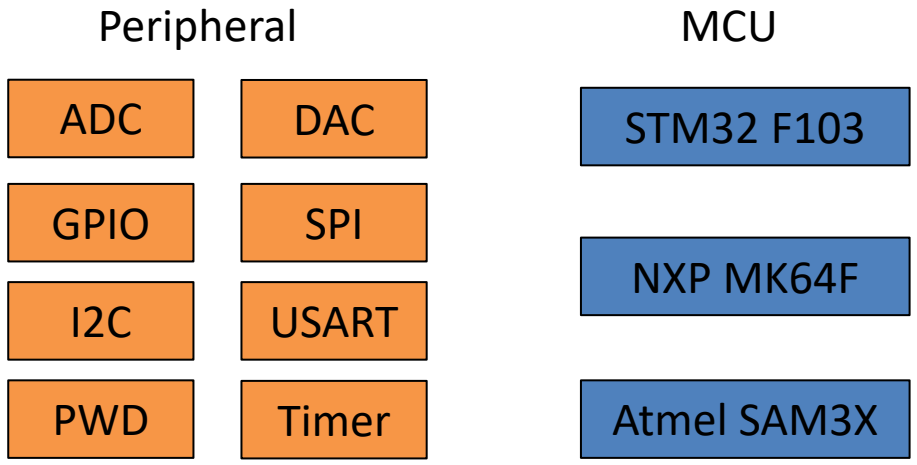
Implementation - Fuzzer Integration



- AFL as a drop-in fuzzer
- Fuzzer Helper
  - Coverage feedback
  - Fork sever
  - Crash/hang detection
  - Data input channel identification

Unit tests

- Reuse the same 66 firmware samples in the P2IM experiment.
  - These samples cover most popular MCU peripherals, MCU chips, MCU OS/system.
- For each unit test, we first run the knowledge extraction phase. During dynamic analysis, we monitor whether its output and execution flow as expected.



OS/Libs

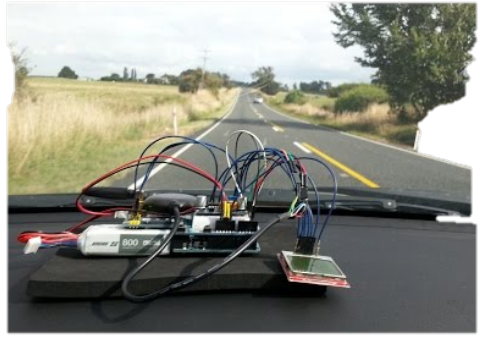
Unit test results

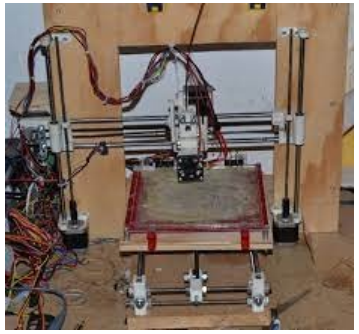
Peripheral	Functional Operations	F103/Arduino	F103/RIOT	F103/NUTTX	K64F/RIOT	SAM3/Arduino	SAM3/RIOT
ADC	Read an analog-to-digital conversion	Pass	N/A	Pass	Pass	Pass	Pass
DAC	Write a value for digital-to-analog conversion	N/A	N/A	N/A	N/A	Pass	Pass
GPIO	Execute callback after pin interrupt	Pass	Pass	Pass	Pass	Pass	Pass
	Read status of a pin	Pass	Pass	Pass	Pass	Pass	Pass
	Set/Clear a pin	Pass	Pass	Pass	Pass	Pass	Pass
PWM	Configure PWM as an autonomous	Pass	N/A	Pass	Pass	Pass	Pass
I2C	Read a byte from a slave	Pass	N/A	**Fail**	**Fail**	Pass	N/A
	Write a byte to a slave	Pass	N/A	N/A	**Fail**	Pass	N/A
UART	Receive a byte	Pass	Pass	Pass	Pass	Pass	Pass
	Transmit a byte	Pass	Pass	Pass	Pass	Pass	Pass
SPI	Receive a byte	Pass	Pass	Pass	Pass	Pass	Pass
	Transmit a byte	Pass	Pass	Pass	Pass	Pass	Pass
Timer	Execute callback after interrupt	N/A	Pass	N/A	Pass	N/A	Pass
	Read counter value	N/A	Pass	N/A	Pass	N/A	Pass

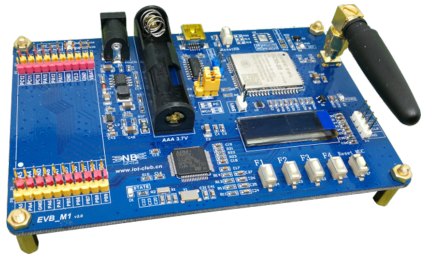
Compared with the passing rate of **79% achieved by state of the art tool P2IM**, µEmu achieves **95% without any manual assistance**. With little manual assistance, all unit tests can be passed.

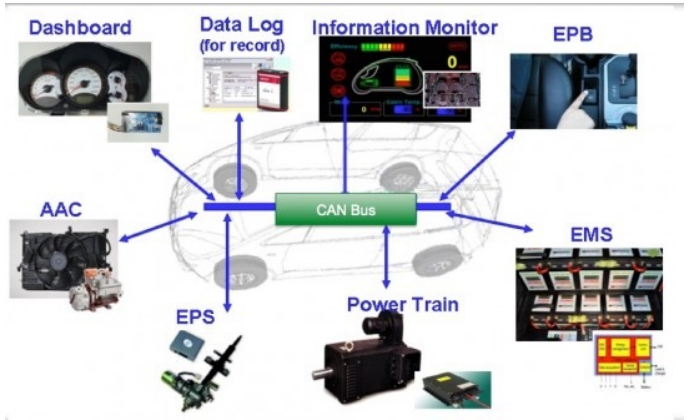
Fuzzing

- Fuzzed 21 real-world firmware
  - 10 from P2IM https://github.com/RiS3-Lab/p2im-real_firmware
  - 3 from HALucinator https://github.com/ucsb-seclab/hal-fuzz
  - 2 from Pretender https://github.com/ucsb-seclab/pretender
  - 6 from more complicate real devices collected by us.









Basic block coverage improvement

- The code coverage increases 10x to 140x compared to that in the normal QEMU without peripheral emulation.

Firmware	BB Cov. Qemu	BB Cov. µEmu	Improv.
CNC	2.68%	67.96%	24.96x
Console	2.19%	35.90%	16.42x
Drone	8.40%	**89.74%**	10.69x
Gateway	1.70%	52.71%	30.94x
Heat Press	1.11%	30.21%	27.22x
Reflow Oven	3.57%	40.53%	11.3x
Robot	2.47%	43.25%	17.51x
Soldering Iron	4.21%	62.01%	14.73x
Steering Control	0.68%	32.59%	48.09x
6LoWPAN Sender	0.88%	48.30%	**55.17x**
6LoWPAN Receiver	0.88%	47.36%	54.08x
RF Door Lock	0.25%	24.37%	**97.57x**
XML Parser	0.72%	26.31%	36.45x
GPS Tracker	0.49%	23.90%	48.81x
…			

Fuzzing results

- Known Bugs Reproduce
- Two previously unknown bugs
  - Steering C: Double Free
  - µtasker_USB: Buffer overflow
- False Crashes/Hangs
  - Due lack of DMA support

Firmware	False Crashes/Hangs	True Crashes/Hang
CNC	0/0	0/0
Drone	0/0	0/0
Gateway	0/0	6/0
Heat_Press	0/0	2/0
PLC	0/0	139/0
Reflow_Oven	0/0	0/0
Soldering_Iron	32/4	0/0
Steering_Control	0/0	12/0
6LoWPAN_Sender	0/0	0/0
6LoWPAN_Receiver	0/0	2/0
RF_Door_Lock	0/0	98/0
Thermostat	0/0	76/0
GPS_Tracker	0/29	29/0
µtasker_USB	0/0	47/0
…		

Future work

- Our current prototype is limit to fuzzing. In the future, we will explore other dynamic approaches on top of uEmu, such as taint analysis.
- We will also Integrate DICE with uEMU to handler DMA.

Summary

- We present µEmu, a security-oriented dynamic analysis platform with the capability to emulate firmware for previously-unseen hardware
- We developed µEmu on top of S2E using plugins and reimplementation of ARM support
- We demonstrate the emulation capability by integrating it with a fuzzer plugin
- We fuzzed 21 real-world firmware and previously-unknown vulnerabilities were found
- Open source at https://github.com/MCUSec/uEmu

Thank You

Questions?

weizhoulightning@gmail.com