Northeastern University

P²IM: Scalable and Hardware-independent Firmware Testing via Automatic Peripheral Interface Modeling

Bo Feng, Alejandro Mera, and Long Lu

Northeastern University

USENIX Security 2020

usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

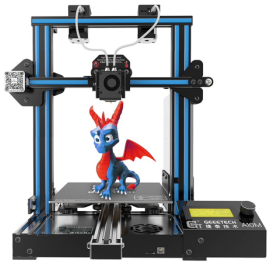
Microcontrollers (MCU) are ubiquitous

Smart light bulb

Fitness tracker

Pulse oximeter

PLC



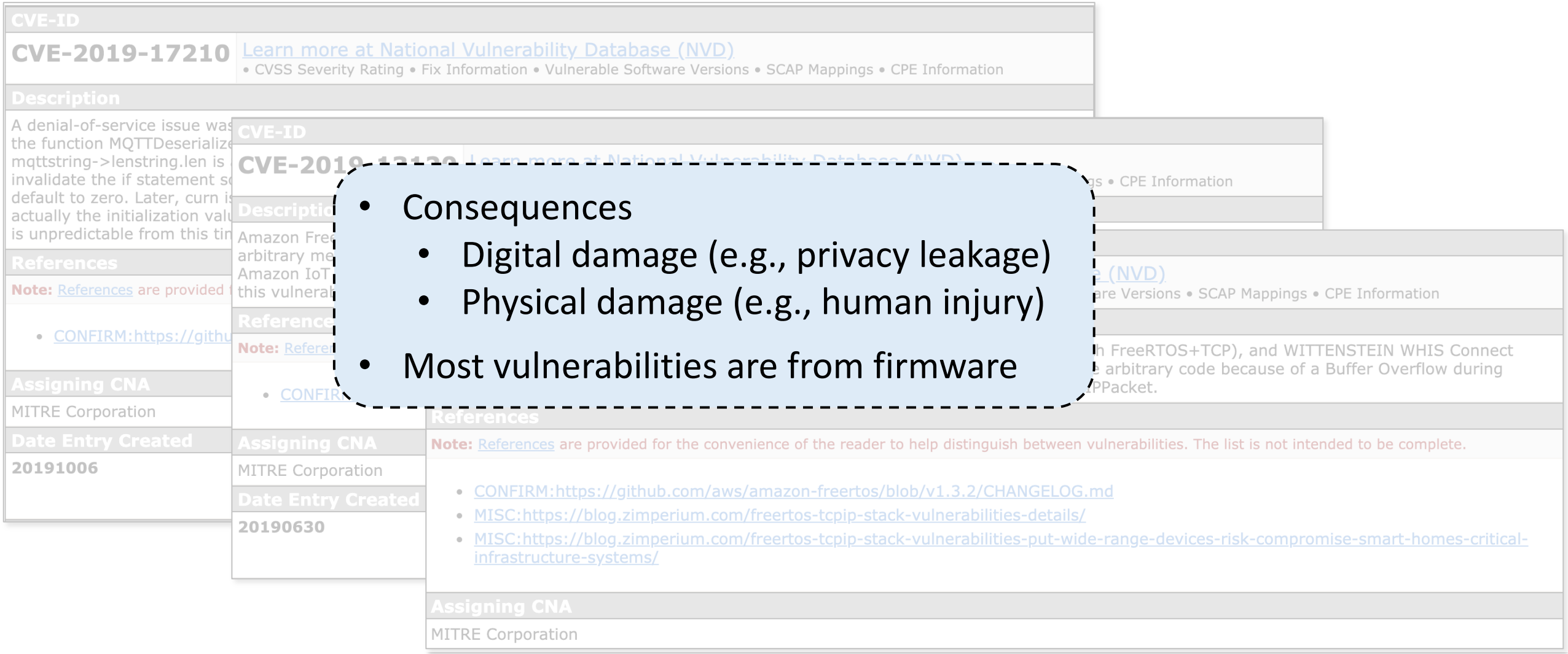
3D printer

Drone

- MCU is a single-chip computer
- 28.1 billion MCUs are sold worldwide in 2018*

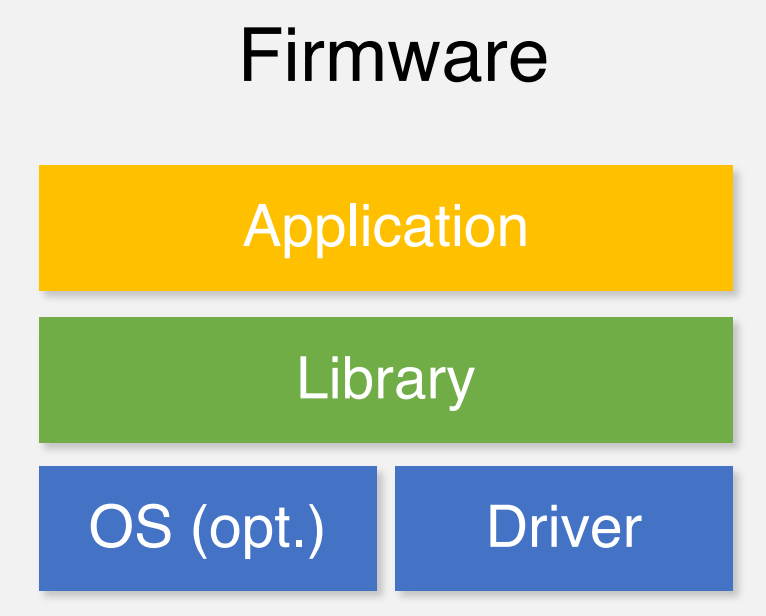
*https://www.statista.com/statistics/935382/worldwide-microcontroller-unit-shipments/

MCU vulnerabilities



- Consequences
  - Digital damage (e.g., privacy leakage)
  - Physical damage (e.g., human injury)
- Most vulnerabilities are from firmware

MCU firmware



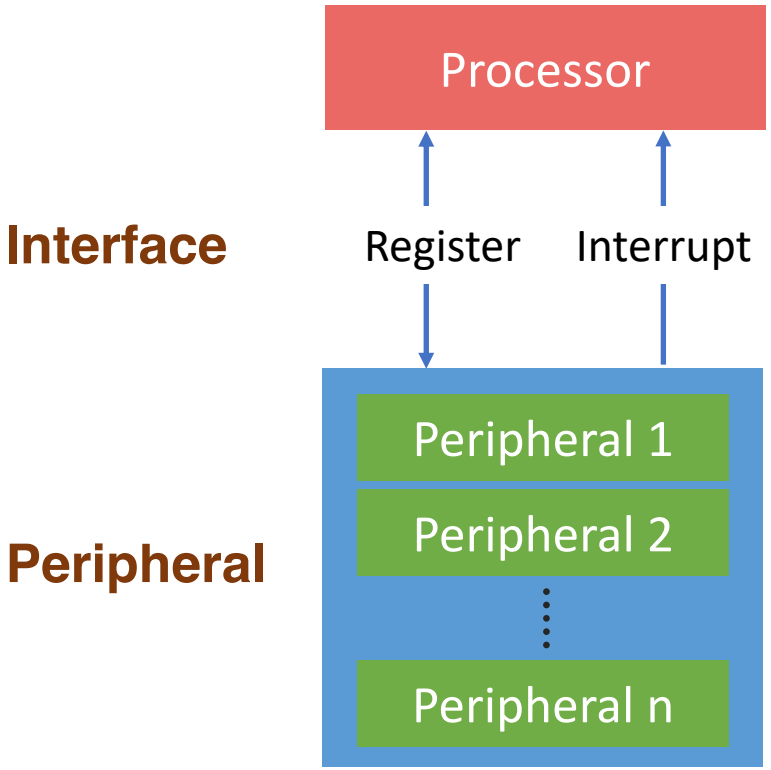
- Whole software stack of the MCU
- Bugs appear in all components

Firmware testing

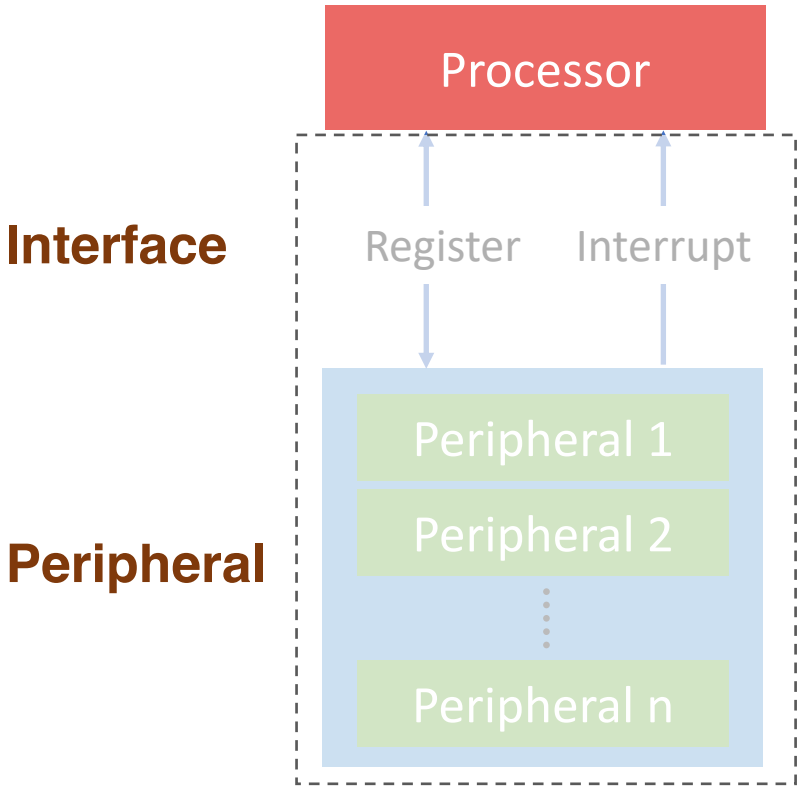
- Fuzzing can effectively find bugs on desktop programs
- As firmware has similar bugs to desktop programs, we test firmware with fuzzers
- Firmware can be tested either on a device or emulator

Because of limited resources on MCU, **on-device fuzzing** is not feasible

Emulator-based firmware testing

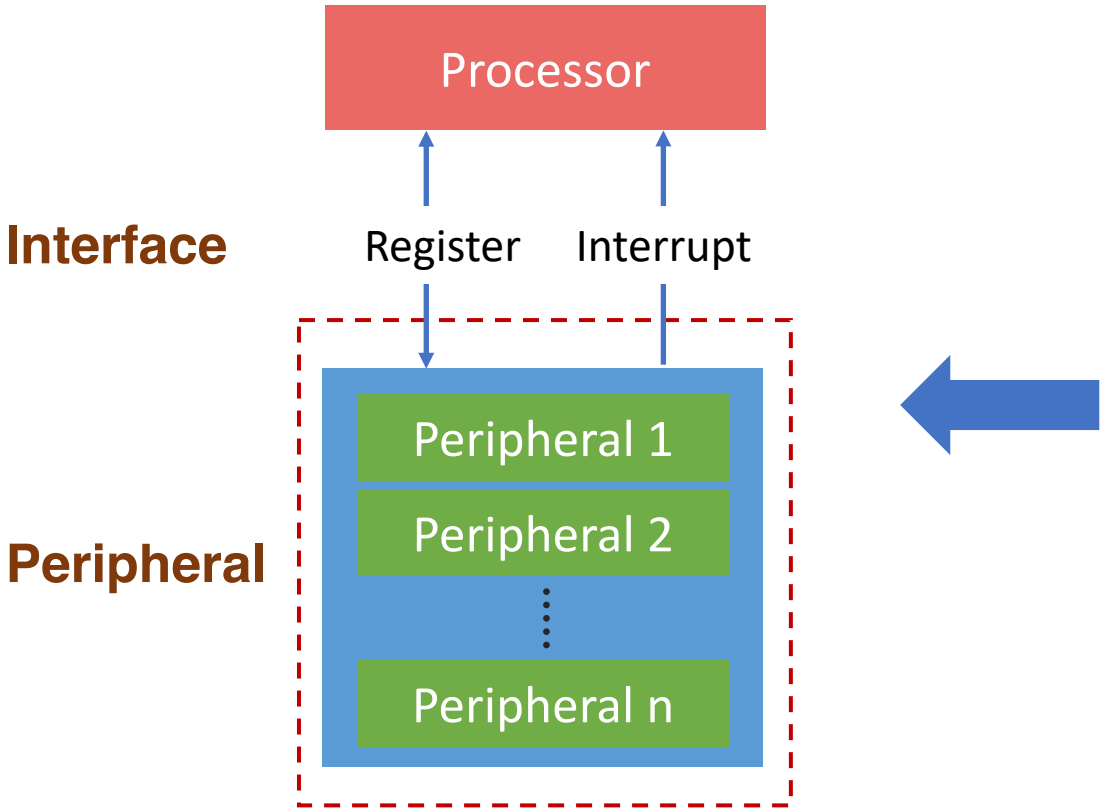


Emulator-based firmware testing



- Not emulated because peripherals are diverse and hard to emulate
- Firmware cannot boot

Existing solution (1)

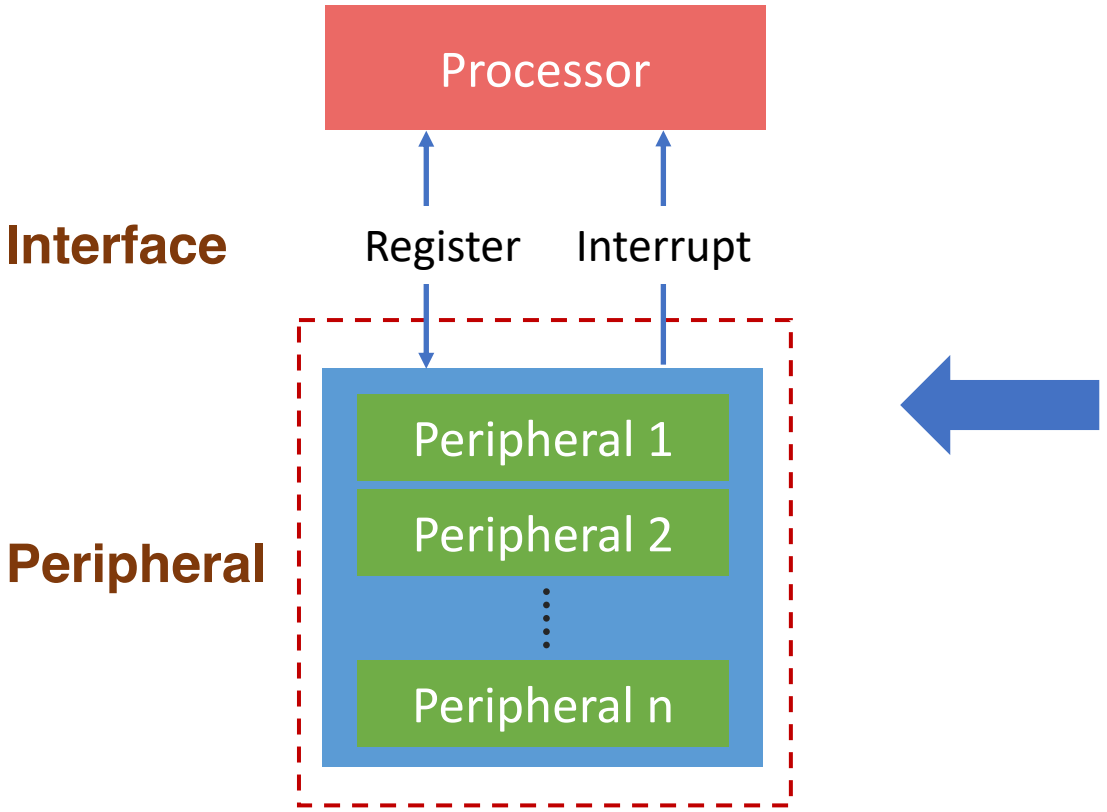


Peripheral emulation:

Emulate peripheral hardware by software components in the emulator

Incomplete support for peripherals, significant manual efforts

Existing solution (2)

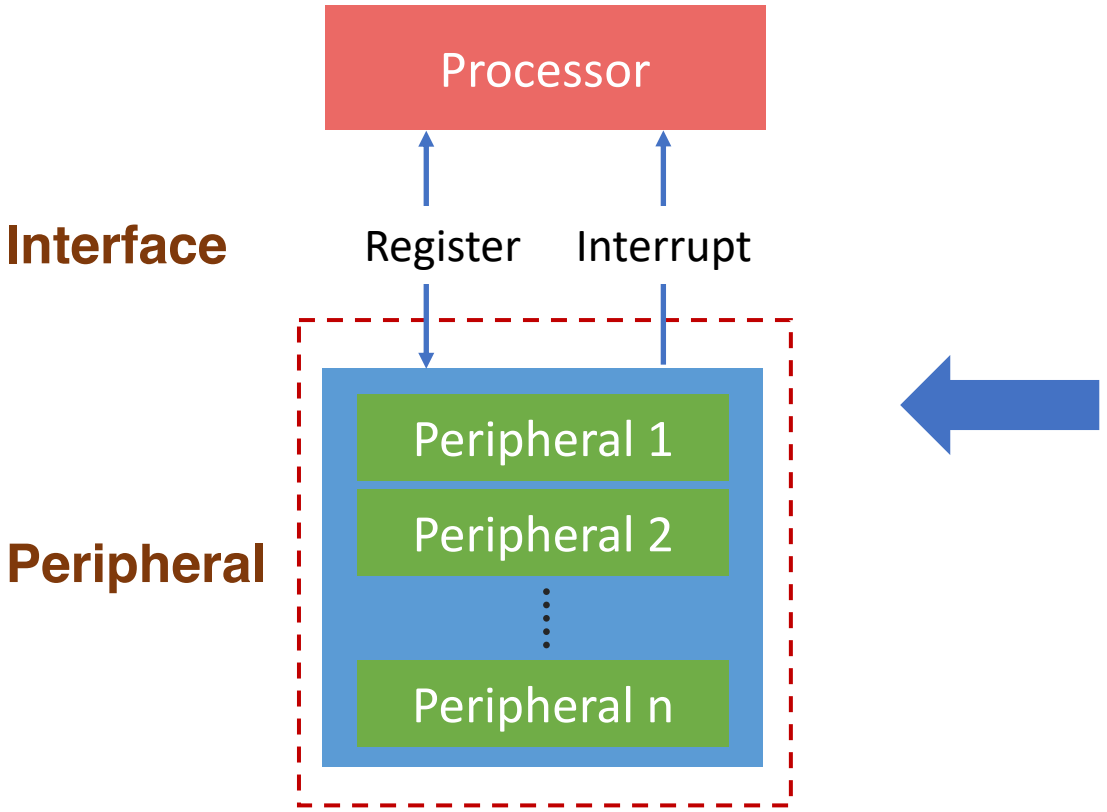


Hardware-in-the-loop emulation:

Use real peripheral hardware to handle peripheral access in the emulator

Rely on real hardware, slow, unscalable

Existing solution (3)



Partial emulation:

Replace peripheral-dependent firmware code with software stubs that have the same functionalities

Unable to test peripheral-dependent code, significant manual efforts

Design goals

Automatic

- A great number of MCU devices need to be tested
- Limited time and money budget for testing
- Human efforts can be minimized

Hardware-independent

- Firmware is tested in the emulator
- Faster and easier to automate

Peripheral-agnostic

- Peripherals are diverse
- Handle peripherals using a uniform approach
- Given a new peripheral, no extra effort is needed

Scalable

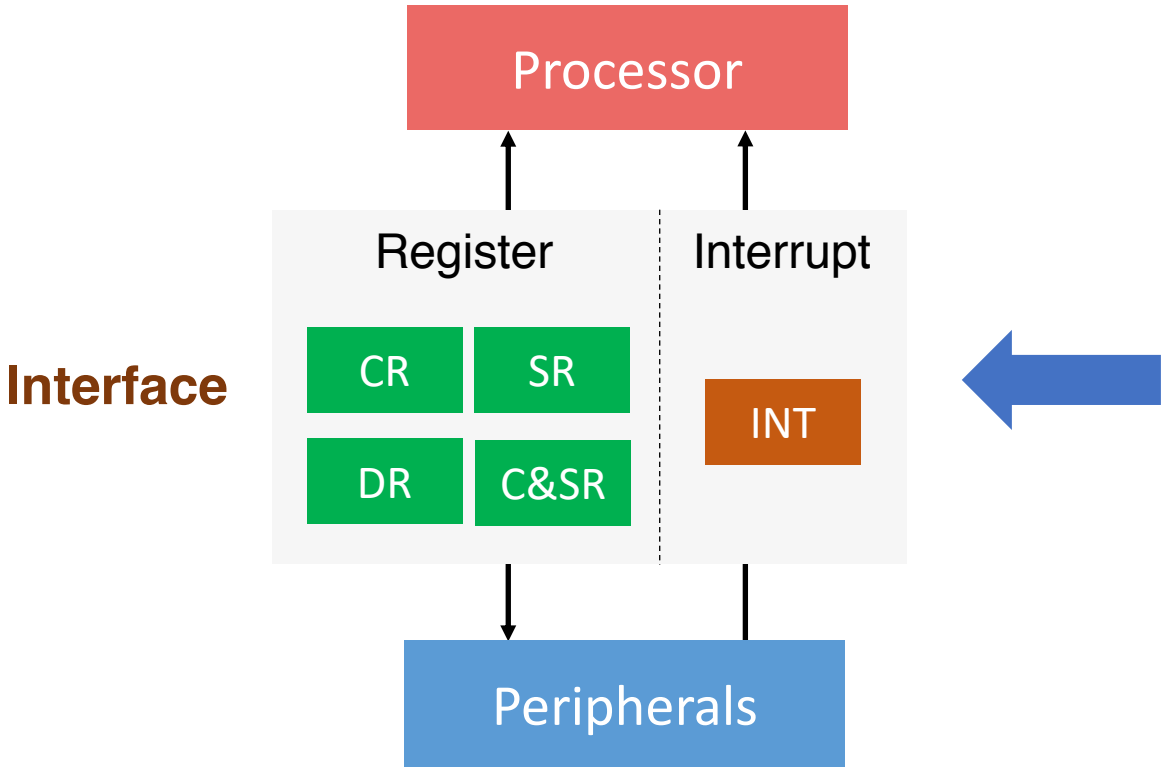
- Multiple fuzzer instances can run in parallel
- Improve code coverage

Observation

- Peripherals are diverse in terms of type and functionality, but interface is not

	Peripheral	Interface
Type	Many	2
Functionality	Many	3
Diversity	**High**	**Low**

Key idea

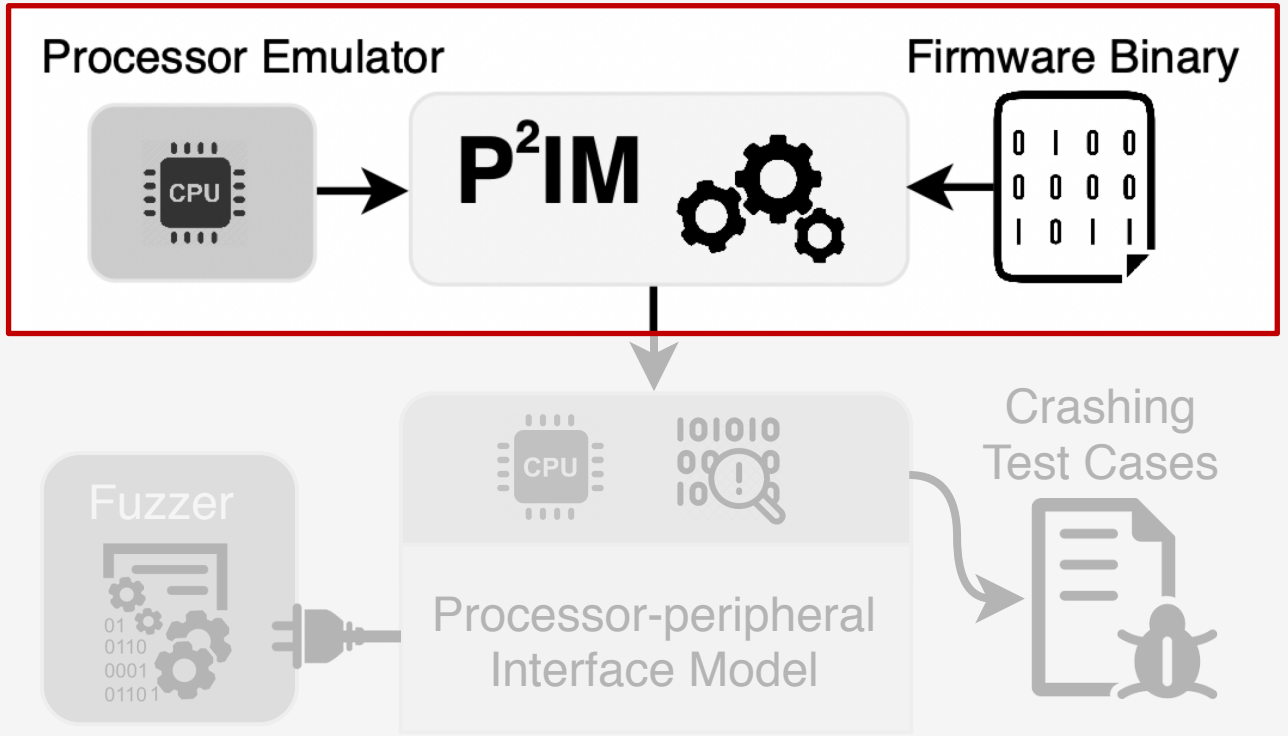


- Treat peripherals as black box
- Abstract a model to handle **register access** and **interrupt firing** for a wide range of peripherals

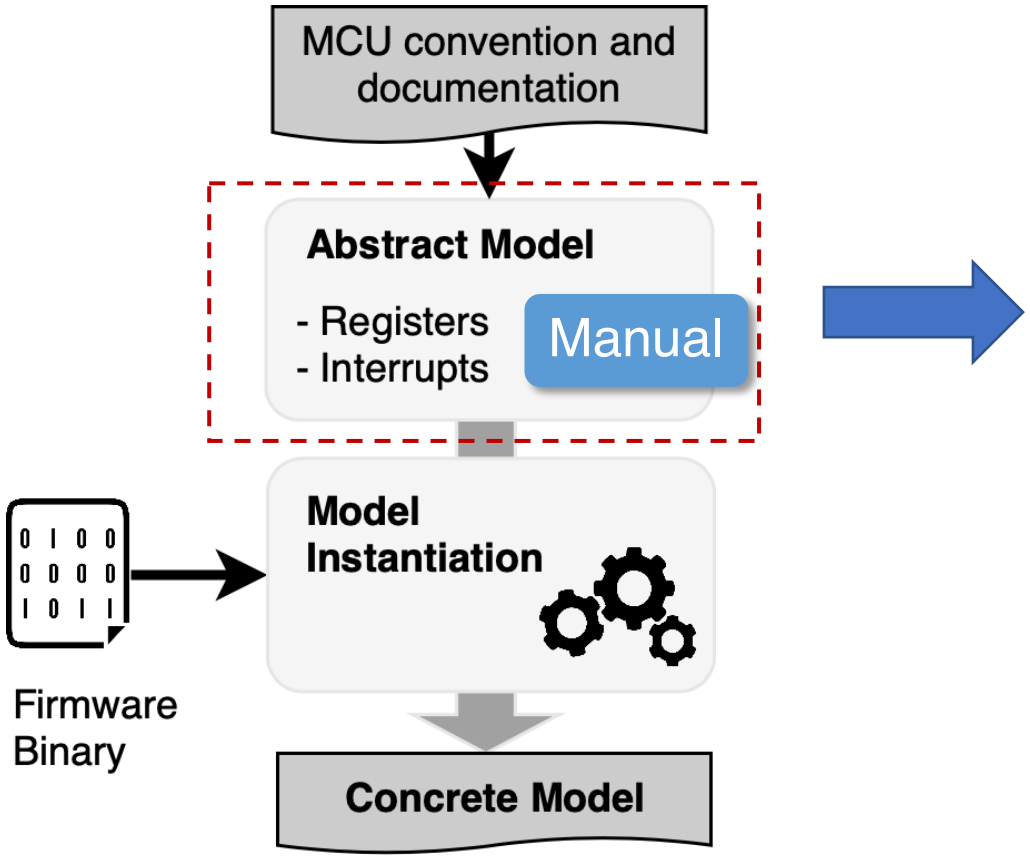
Comparison with state-of-the-art

Approaches	Automatic	Hardware-independent	Peripheral-agnostic	Scalable	Existing work
Peripheral emulation	x	✓	x	✓	GNU MCU Eclipse QEMU (2015), PartEmu (Usenix '20)
Hardware-in-the-loop emulation	x	x	✓	x	Avatar (NDSS '14), Prospect (Asia CCS '14), Surrogates (WOOT '15), Charm (Usenix '18)
Partial emulation	x	✓	✓	✓	Firmadyne (NDSS '16), HALucinator (Usenix '20), PartEmu (Usenix '20)
P²IM (our work)	✓	✓	✓	✓	

Workflow


Processor Emulator
Firmware Binary
CPU
P²IM
CPU
Fuzzer
Processor-peripheral
Interface Model
Crashing
Test Cases

Interface modeling



Peripherals determine register value and interrupt-firing timing, but peripherals are considered as black box

How to model an interface?

Registers are categorized by their functionalities and handled accordingly

Interrupts can be fired at any time. We use a fixed frequency

Register categories

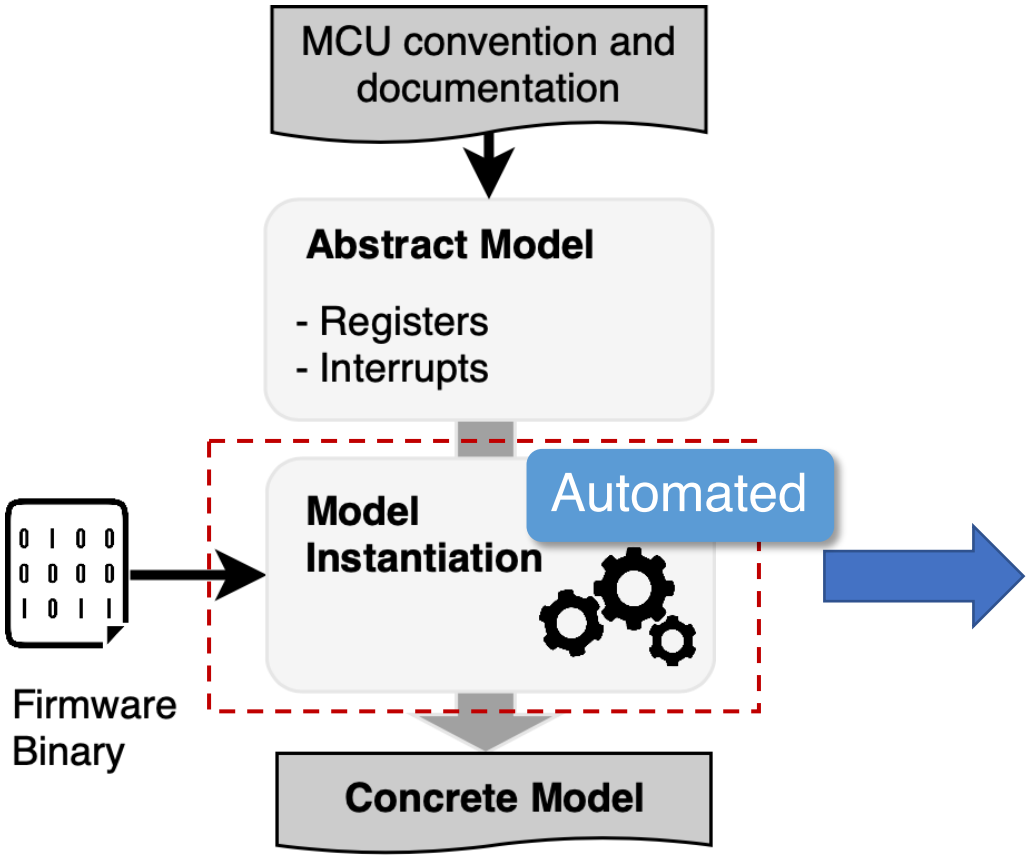
Control register (CR)

Status register (SR)

Data register (DR)

Control-status register (C&SR)

Interface modeling (2)

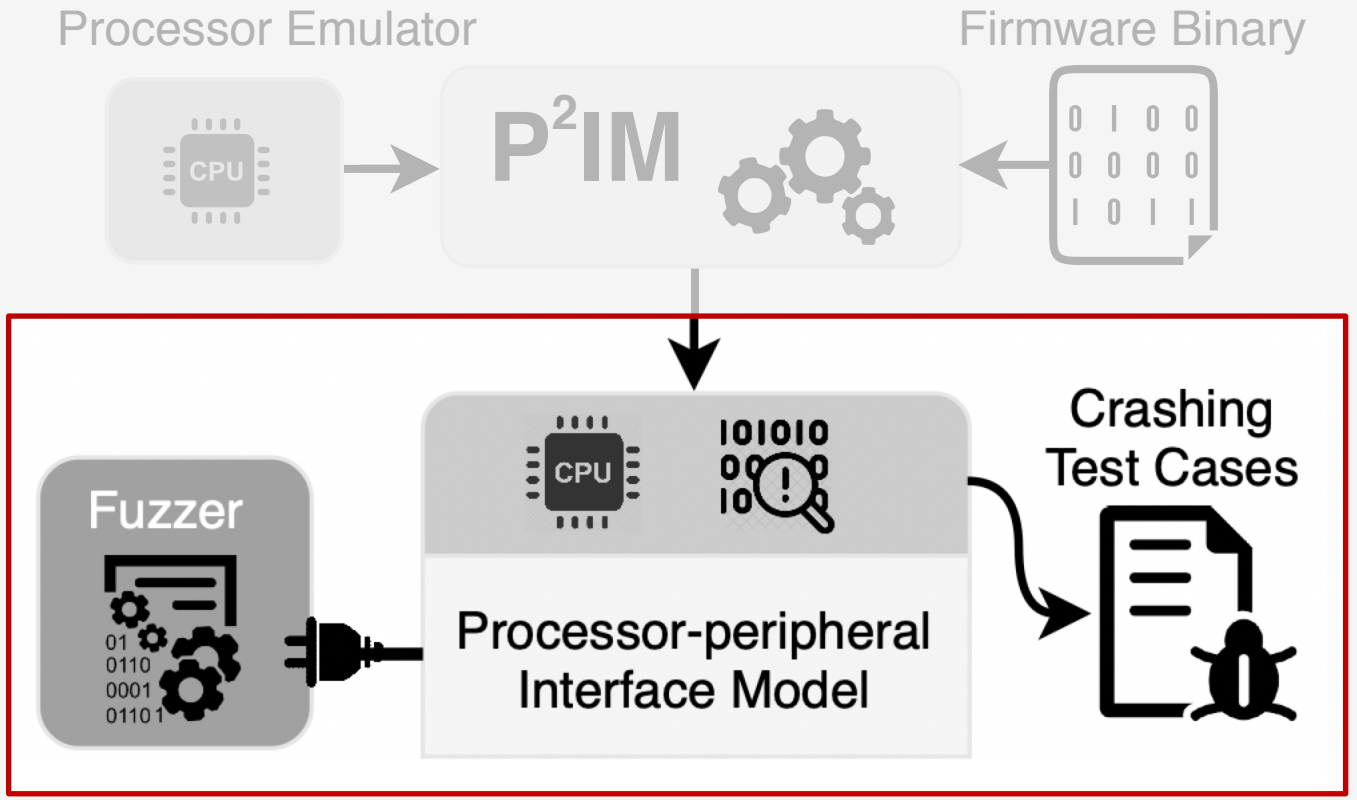


Given a firmware, how to identify the interface needs to be modeled?

Registers are identified and categorized by monitoring access to the memory-mapped peripheral region

Interrupts are detected by monitoring the interrupt controller

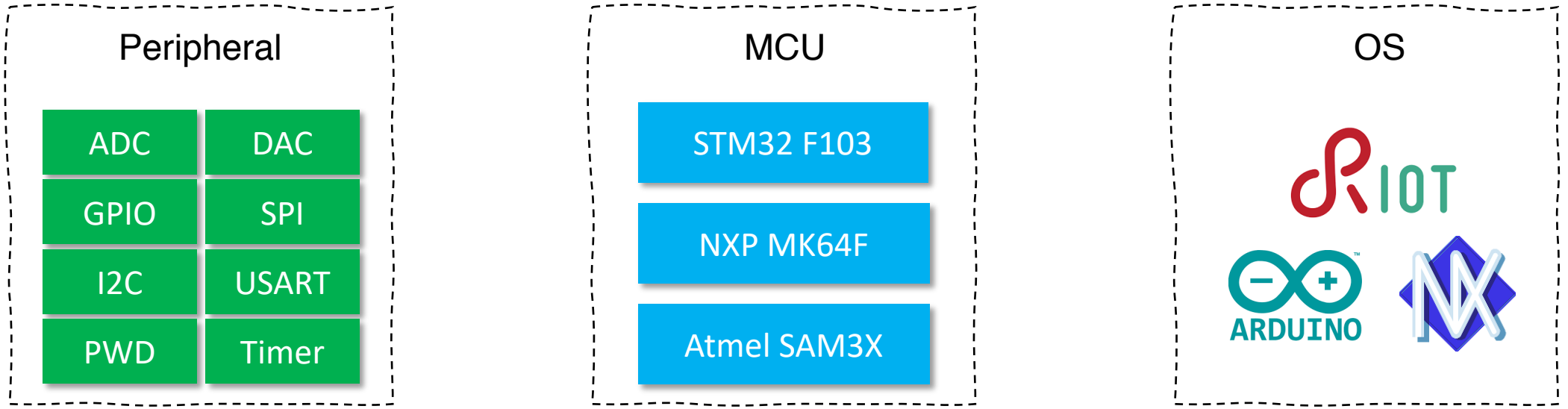
Workflow



Existing fuzzers can be used without modification

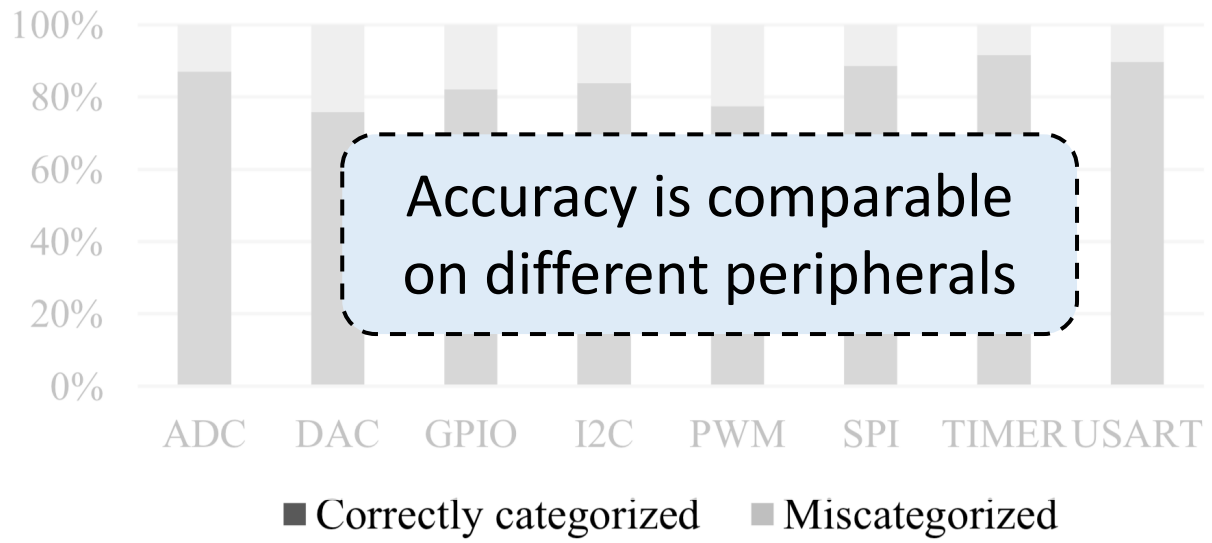
Evaluation

- 70 sample firmware for essential peripheral operations
  - E.g., data transmission through USART peripheral



Results

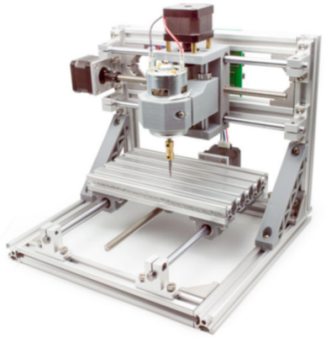
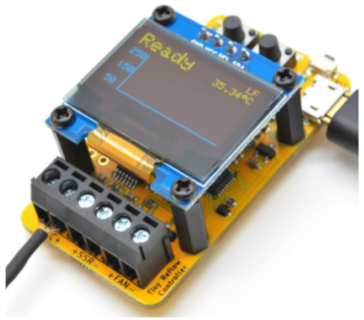
- The majority of firmware boot and perform essential peripheral operations normally
  - 79% test cases pass
- The accuracy of register categorization is between 76% and 92%



Fuzzing

- Fuzz-test 10 real-world firmware
  - Drone, Robot, Gateway, PLC, etc.



Fuzzing performance

- The accuracy of register categorization is between 69.6% and 100%
- Speed and basic block coverage:

Firmware	Speed (# tests/s)	Basic block coverage	Coverage improvement
Drone	17.2	58%	7x
CNC	18.0	70%	26x
Steering C.	32.3	20%	30x
…			

Fuzzing result

- Detect 7 unique bugs, all of which are
  - Previously unknown
  - Remotely exploitable
  - Reproducible on real device

Firmware	Unique bugs	Bug nature
PLC	3	Incorrect Type Cast
	1	Integer overflow
	1	Incorrect Conversion between Numeric Types
Gateway	1	Buffer overflow
Heat Press	1	Buffer overflow

Summary

- Propose **P²IM**, the **first** scalable and hardware-independent firmware testing framework
- Design and implement a novel interface modeling mechanism
- Fuzz-test 10 real-world firmware
- Find 7 previously-unknown vulnerabilities

Code and Tested Firmware at:

https://github.com/RiS3-Lab/p2im

Thank You

Questions?

 feng.bo@northeastern.edu