FirmGuide: Boosting the Capability of Rehosting Embedded Linux Kernels through Model-Guided Kernel Execution

Qiang Liu[1*] Cen Zhang[2*] Lin Ma[1] Muhui Jiang[1,3] Yajin Zhou[1] Lei Wu[1] Wenbo Shen[1] Xiapu Luo[3] Yang Liu[2] Kui Ren[1]

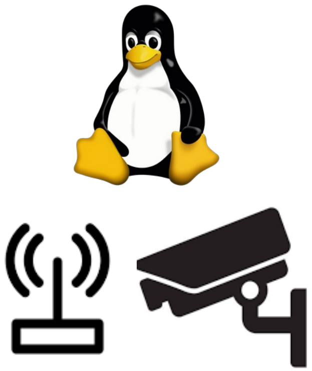
[1]Zhejiang University [2]Nanyang Technological University [3]The Hong Kong Polytechnic University

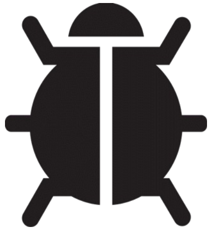
*The first two authors contributed equally to this work.

ASE2021

Motivation

Dynamic Bug or Vulnerability Understanding

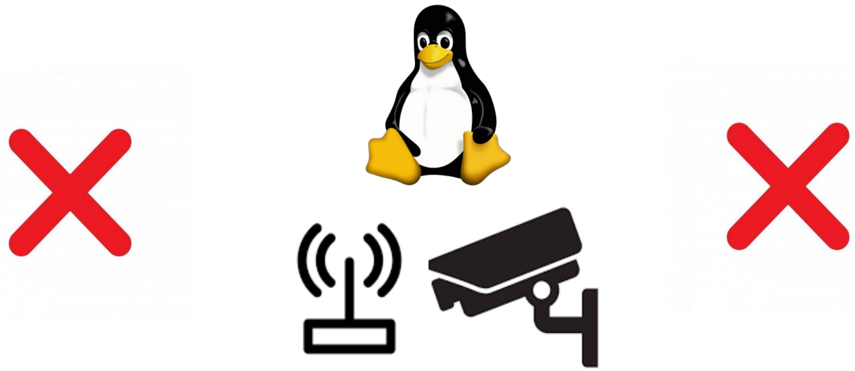


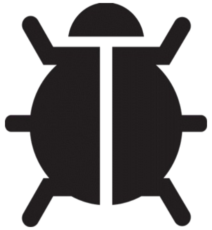


Dynamic Bug or Vulnerability Mining

Motivation

Dynamic Bug or Vulnerability Understanding

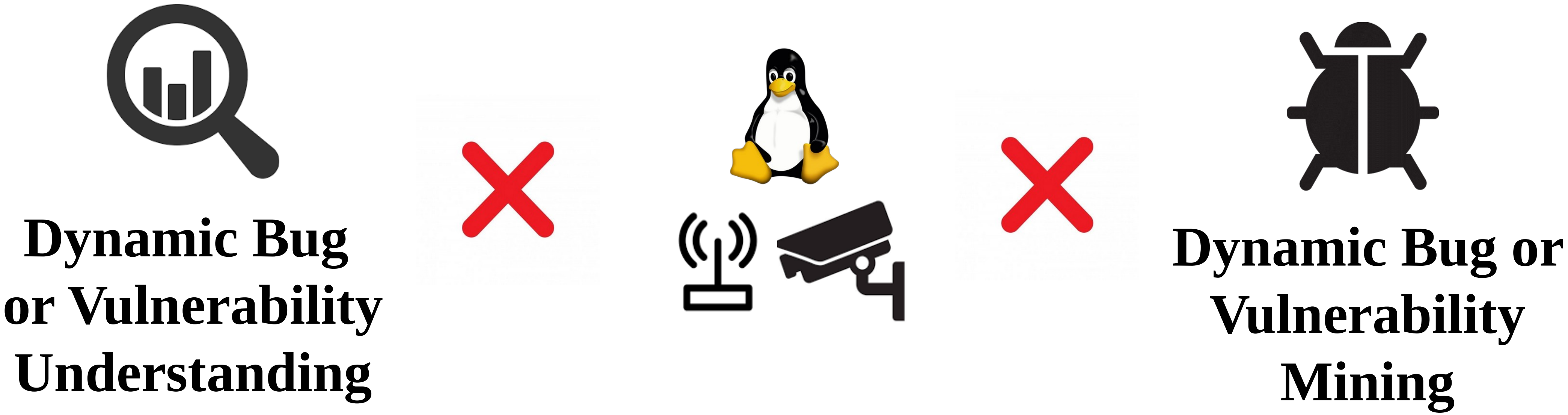




Dynamic Bug or Vulnerability Mining

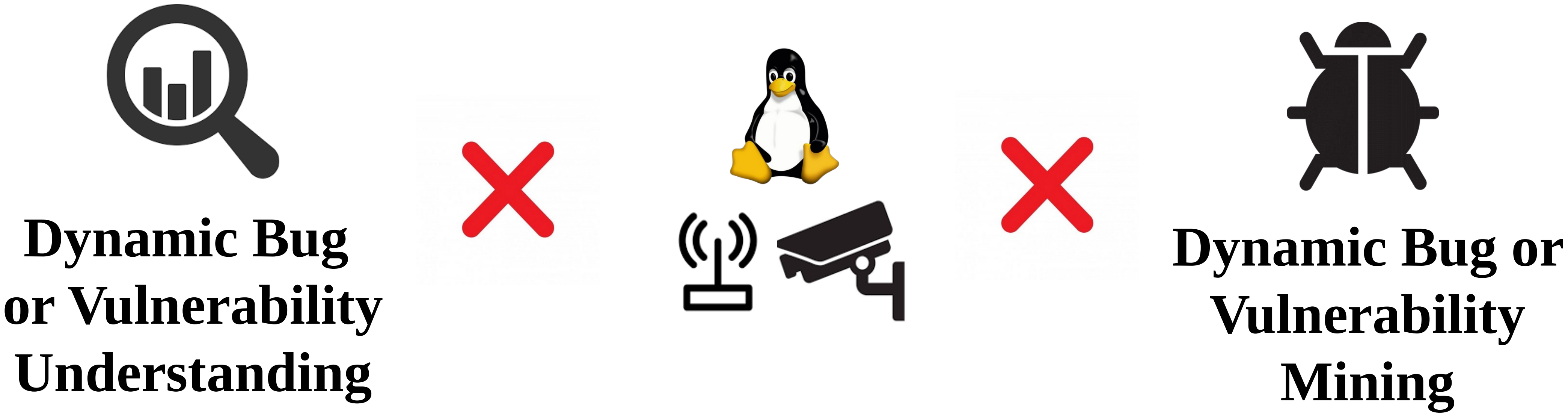
- Linux kernel with drivers inside high-end embedded firmware

Motivation



- Linux kernel with drivers inside high-end embedded firmware
- Understanding and testing abilities not easy and scaling due to hardware requirement

Motivation



- Linux kernel with drivers inside high-end embedded firmware
- Understanding and testing abilities not easy and scaling due to hardware requirement
- **Rehosting the embedded Linux kernel with the best effort**

Challenge and Observation 1

SoC: plxtech,nas782x	
CPU	Arm11MPCore
Memory	up to 512M
Interrupt Controller	gic
Time-related	rps, oscillator, sysclk, plla, pllb, stdclk, twdclk
UART	ns16550a
Others	gmacclk, pcie, watchdog, sata, nand, ethernet, ehci, leds

Challenge and Observation 1

SoC: plxtech,nas782x	
CPU	Arm11MPCore
Memory	up to 512M
Interrupt Controller	gic
Time-related	rps, oscillator, sysclk, plla, pllb, stdclk, twdclk
UART	ns16550a
Others	gmacclk, pcie, watchdog, sata, nand, ethernet, ehci, leds

- Numerous peripherals: Type-I Type-II
- **Classifying peripherals for a minimum best effort**

Challenge and Observation 1

SoC: plxtech,nas782x	
CPU	Arm11MPCore
Memory	up to 512M
Interrupt Controller	gic
Time-related	rps, oscillator, sysclk, plla, pllb, stdclk, twdclk
UART	ns16550a
Others	gmacclk, pcie, watchdog, sata, nand, ethernet, ehci, leds

High fidelity to make the Linux kernel functional-correct

Low fidelity for successful boot

- Numerous peripherals: Type-I High Fidelity Type-II Low Fidelity
- **Classifying peripherals for a minimum best effort**

Challenge and Observation 1

SoC: plxtech,nas782x	
CPU	Arm11MPCore
Memory	up to 512M
Interrupt Controller	gic
Time-related	rps, oscillator, sysclk, plla, pllb, stdclk, twdclk
UART	ns16550a
Others	gmacclk, pcie, watchdog, sata, nand, ethernet, ehci, leds

High fidelity to make the Linux kernel functional-correct

Low fidelity for successful boot

High-fidelity Virtual Device

- Numerous peripherals: Type-I High Fidelity Type-II Low Fidelity
- **Classifying peripherals for a minimum best effort**

Challenge and Observation 1

SoC: plxtech,nas782x	
CPU	Arm11MPCore
Memory	up to 512M
Interrupt Controller	gic
Time-related	rps, oscillator, sysclk, plla, pllb, stdclk, twdclk
UART	ns16550a
Others	gmacclk, pcie, watchdog, sata, nand, ethernet, ehci, leds

High fidelity to make the Linux kernel functional-correct

Low fidelity for successful boot

- Numerous peripherals: Type-I High Fidelity Type-II Low Fidelity
- **Classifying peripherals for a minimum best effort**

High-fidelity Virtual Device

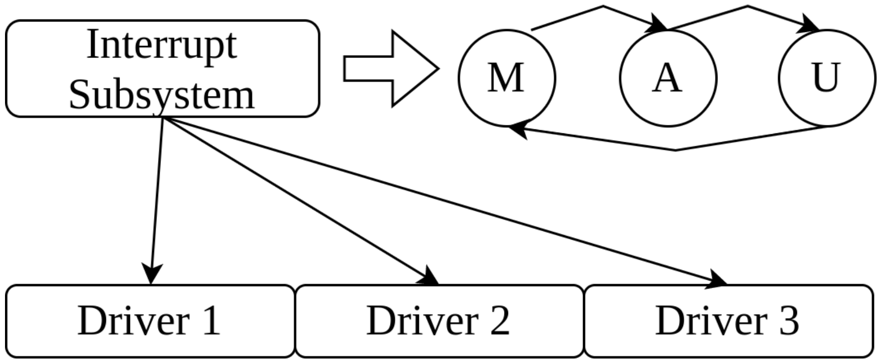
Dummy Virtual Device

Challenge and Observation 2

Challenge and Observation 2

Multiple models for interrupt controllers

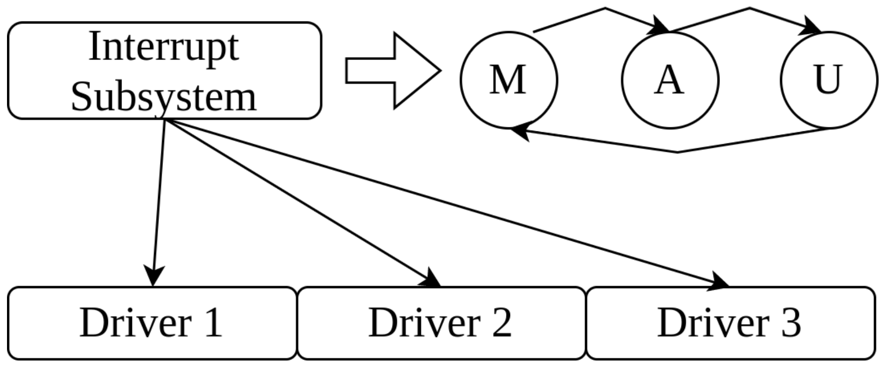
- ralink-rt2880-intc
- qca,ar7240-intc
- marvell,orion-intc
- marvell,orion-bridge-intc
- arm,cortex-a9-gic
- …



Challenge and Observation 2

Multiple models for interrupt controllers

- ralink-rt2880-intc
- qca,ar7240-intc
- marvell,orion-intc
- marvell,orion-bridge-intc
- arm,cortex-a9-gic
- …

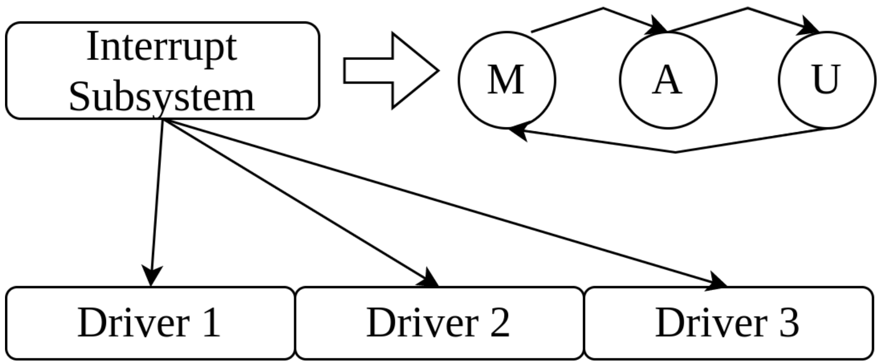


- Diverse models: Linux subsystems that hide implementation details

Challenge and Observation 2

Multiple models for interrupt controllers

ralink-rt2880-intc
qca,ar7240-intc
marvell,orion-intc
marvell,orion-bridge-intc
arm,cortex-a9-gic
…

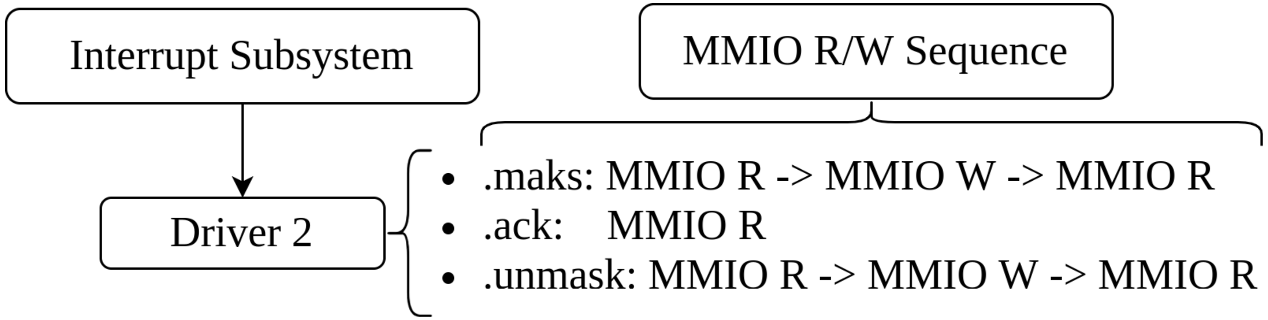


- Diverse models: Linux subsystems that hide implementation details
- **Extracting state machines from the Linux subsystems (Type-I)**

Challenge and Observation 3

Challenge and Observation 3

```
Mask Interrupt
    MMIO Read M -> a
    a &= flags
    MMIO Write a -> M
Load IRQ number
    MMIO Read I -> b
    switch(b)
        …
```



Challenge and Observation 3

Mask Interrupt

    MMIO Read M -> a

    a &= flags

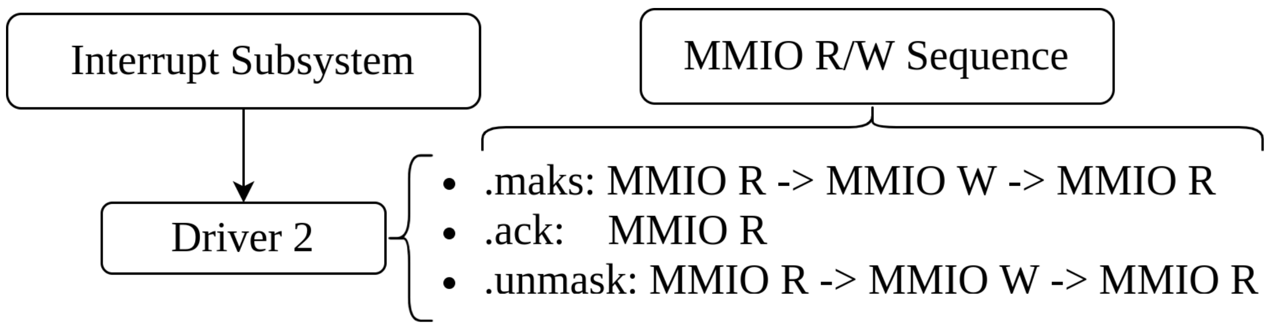
    MMIO Write a -> M

Load IRQ number

    MMIO Read I -> b

    switch(b)

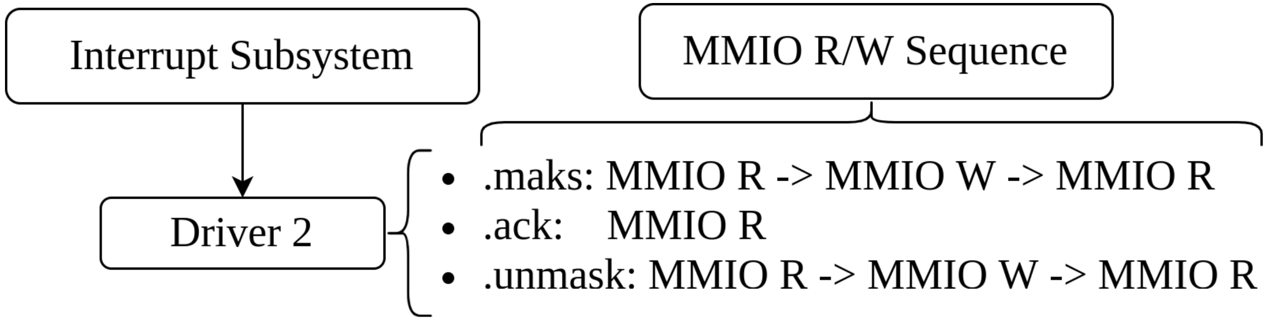
        …



- Complex semantics: Specific driver interface callbacks that embed complex semantics

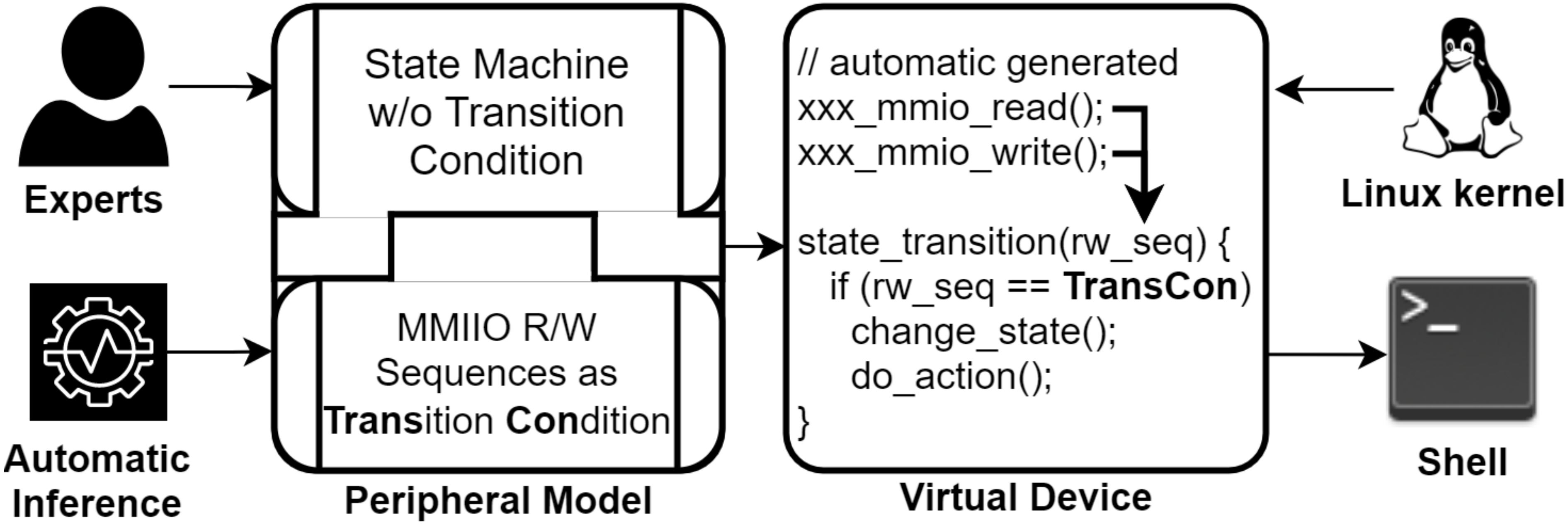
Challenge and Observation 3

Mask Interrupt
    MMIO Read M -> a
    a &= flags
    MMIO Write a -> M
Load IRQ number
    MMIO Read I -> b
    switch(b)
        …



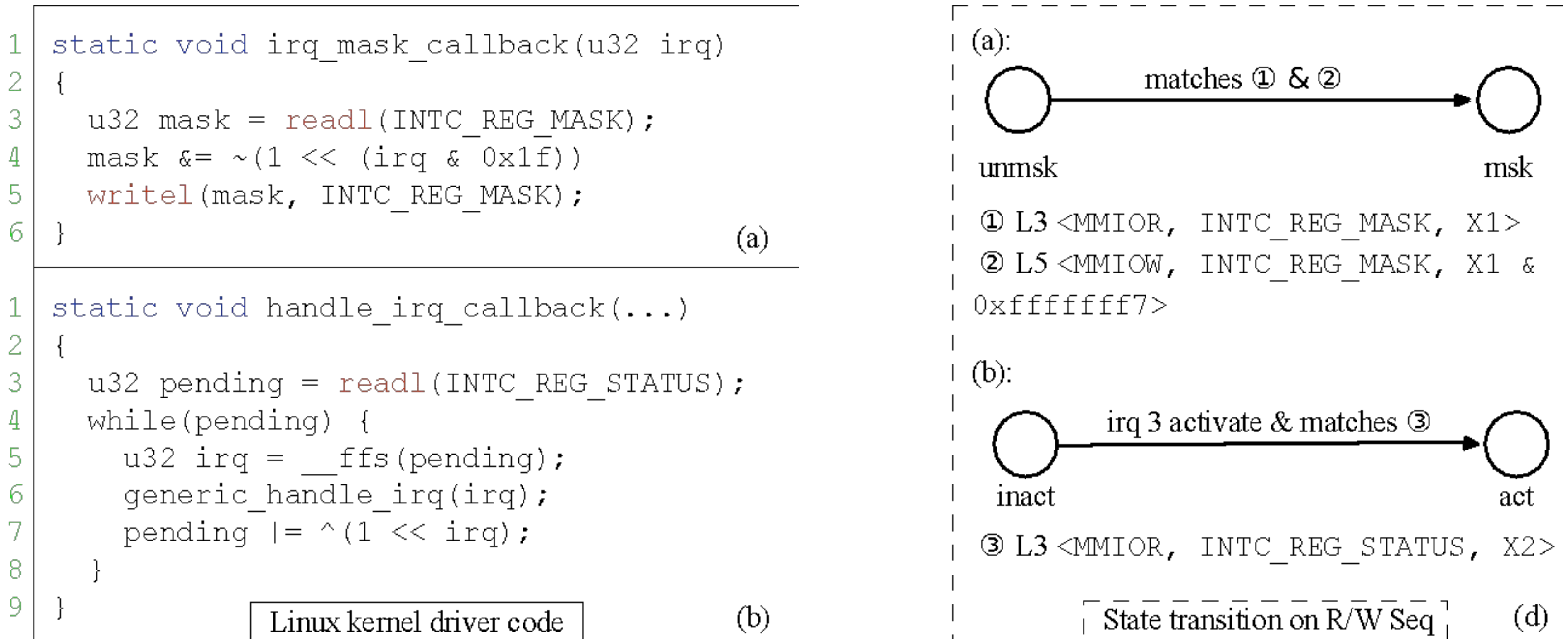
- Complex semantics: Specific driver interface callbacks that embed complex semantics
- **Extracting MMIO R/W sequences from these callbacks (Type-I)**

Core Technique: Model-guided Kernel Execution



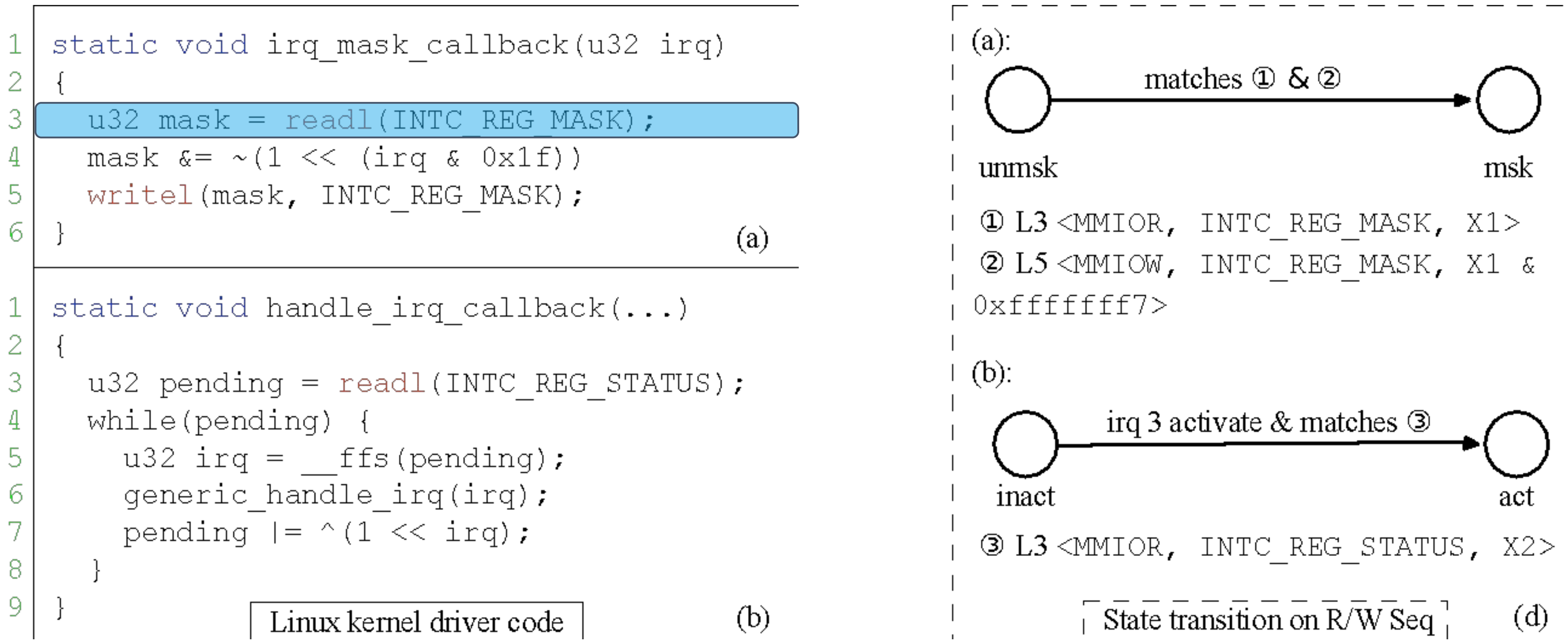
- Peripheral model = the model template (a state machine) + the model parameters (MMIO R/W sequences as transition conditions)

Model-guided Kernel Execution: Running Example



- **The MMIO Read/Write sequence from Linux kernel can be recognized to drive the state machine of our emulated peripherals**

Model-guided Kernel Execution: Running Example



- **The MMIO Read/Write sequence from Linux kernel can be recognized to drive the state machine of our emulated peripherals**

Model-guided Kernel Execution: Running Example

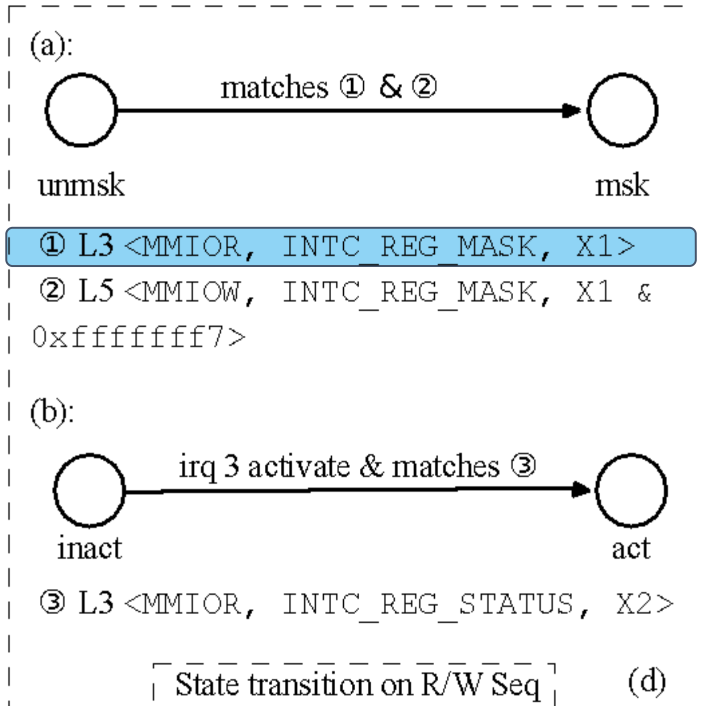
```
1 static void irq_mask_callback(u32 irq)
2 {
3   u32 mask = readl(INTC_REG_MASK);
4   mask &= ~(1 << (irq & 0x1f))
5   writel(mask, INTC_REG_MASK);
6 }
```

(a)

```
1 static void handle_irq_callback(...)
2 {
3   u32 pending = readl(INTC_REG_STATUS);
4   while(pending) {
5     u32 irq = __ffs(pending);
6     generic_handle_irq(irq);
7     pending |= ^(1 << irq);
8   }
9 }
```

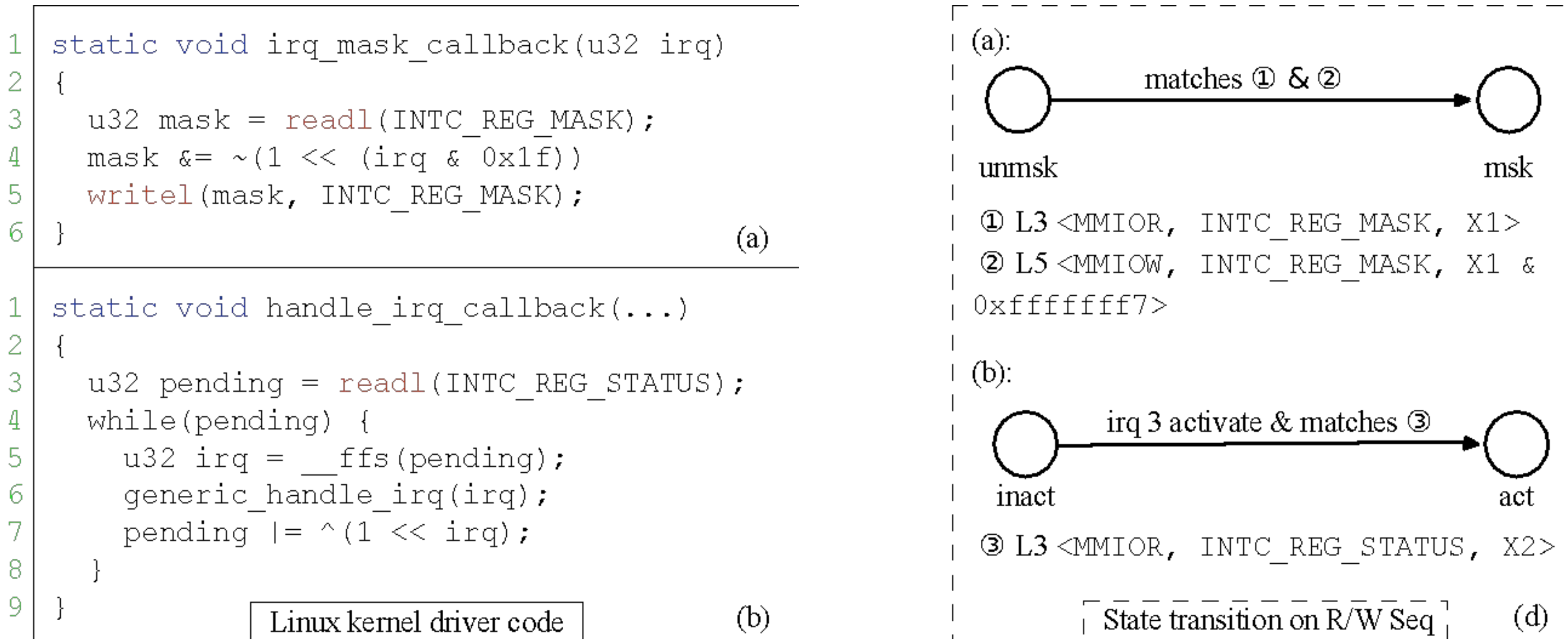
Linux kernel driver code

(b)



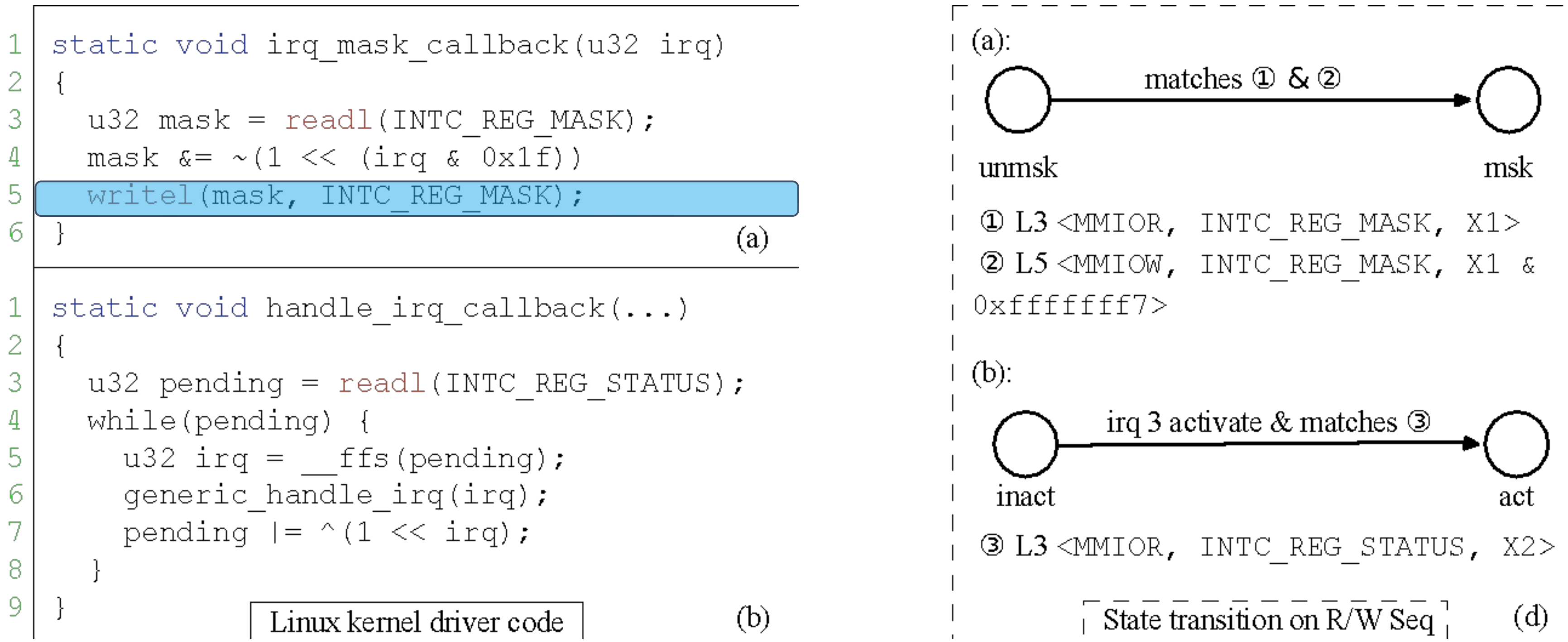
- **The MMIO Read/Write sequence from Linux kernel can be recognized to drive the state machine of our emulated peripherals**

Model-guided Kernel Execution: Running Example



- **The MMIO Read/Write sequence from Linux kernel can be recognized to drive the state machine of our emulated peripherals**

Model-guided Kernel Execution: Running Example



- **The MMIO Read/Write sequence from Linux kernel can be recognized to drive the state machine of our emulated peripherals**

Model-guided Kernel Execution: Running Example

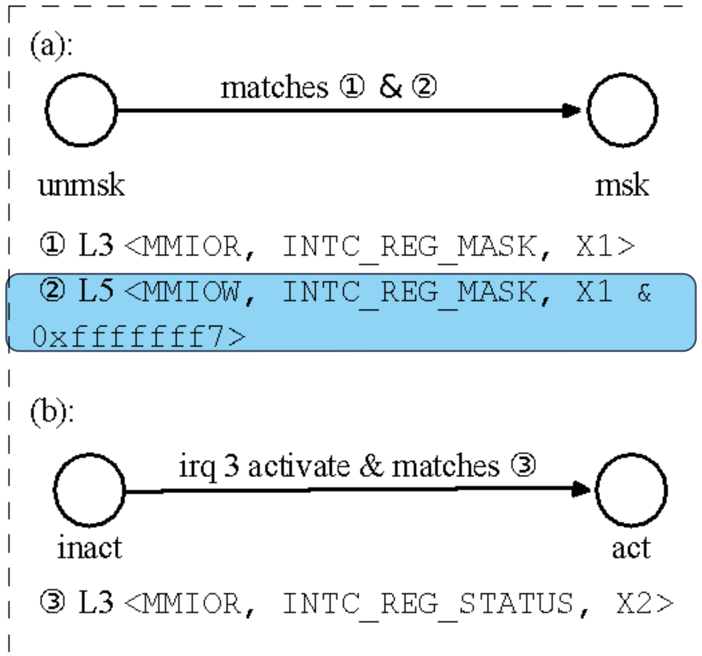
```
1 static void irq_mask_callback(u32 irq)
2 {
3   u32 mask = readl(INTC_REG_MASK);
4   mask &= ~(1 << (irq & 0x1f))
5   writel(mask, INTC_REG_MASK);
6 }
```

(a)

```
1 static void handle_irq_callback(...)
2 {
3   u32 pending = readl(INTC_REG_STATUS);
4   while(pending) {
5     u32 irq = __ffs(pending);
6     generic_handle_irq(irq);
7     pending |= ^(1 << irq);
8   }
9 }
```

Linux kernel driver code

(b)

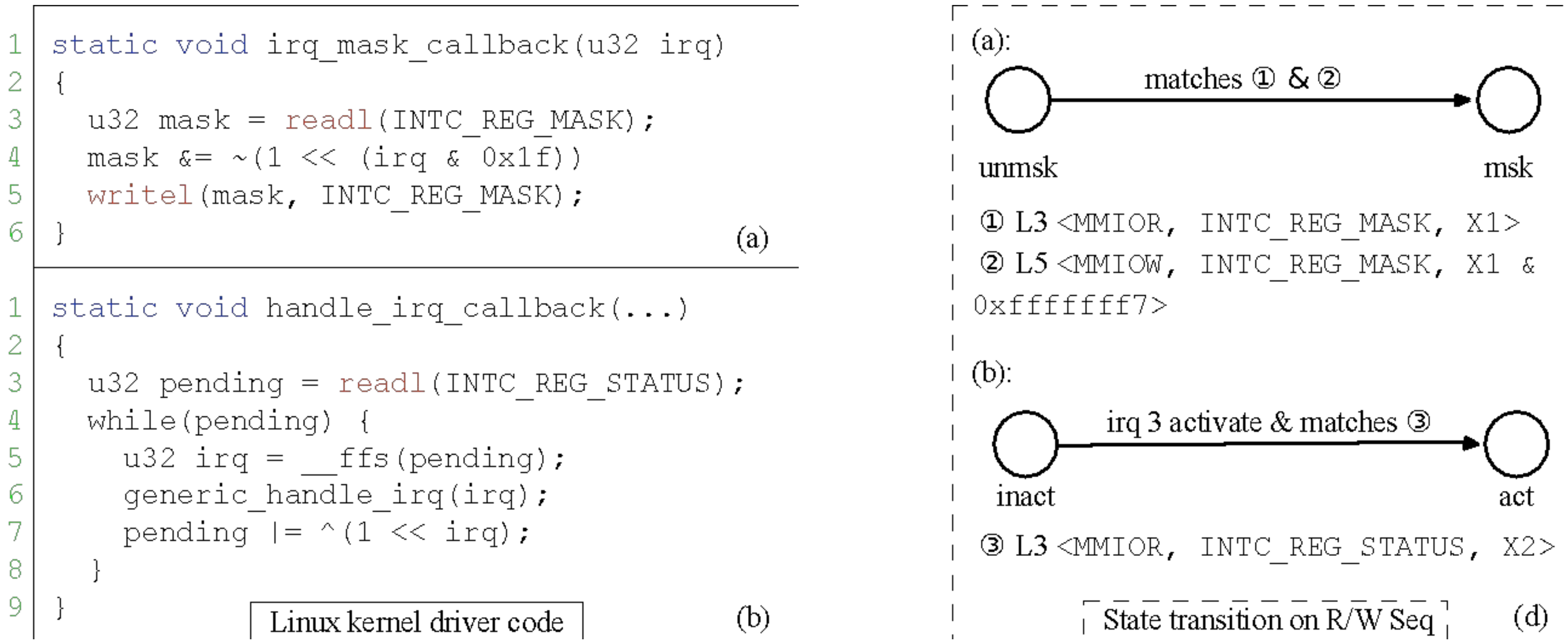


State transition on R/W Seq

(d)

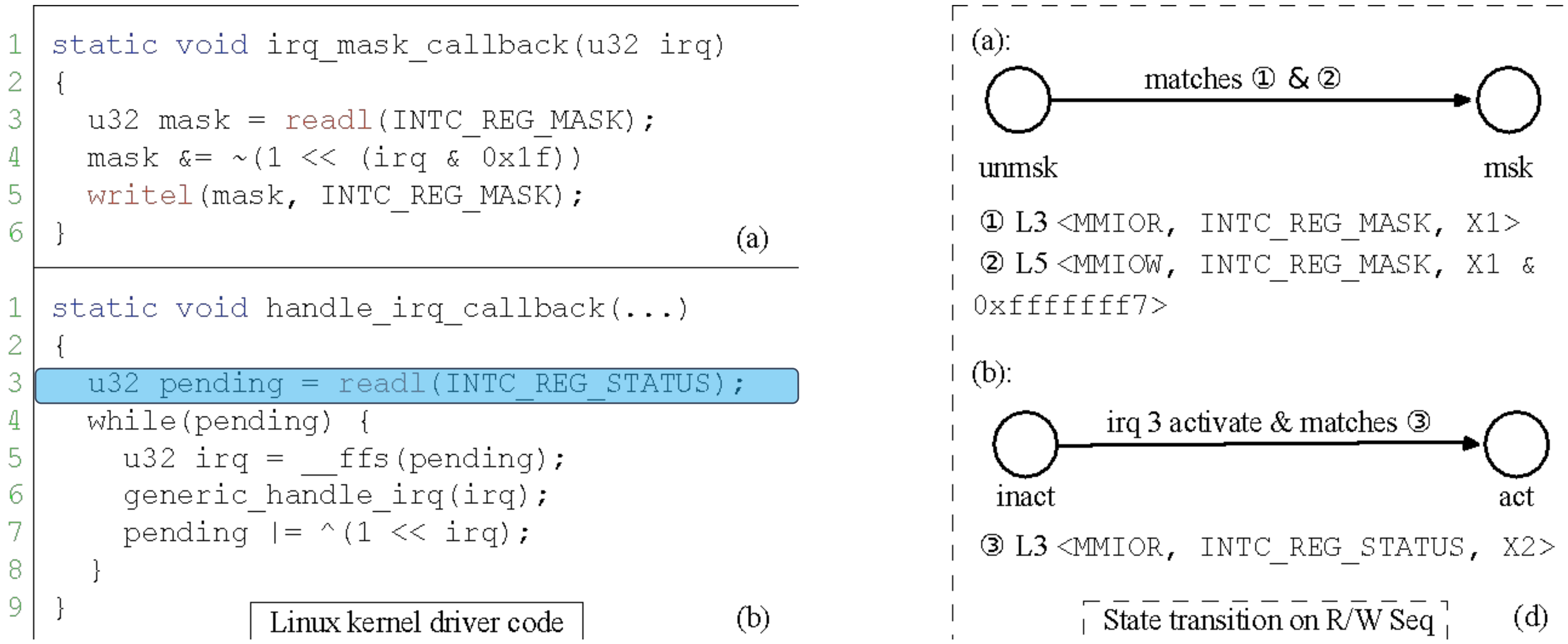
- **The MMIO Read/Write sequence from Linux kernel can be recognized to drive the state machine of our emulated peripherals**

Model-guided Kernel Execution: Running Example



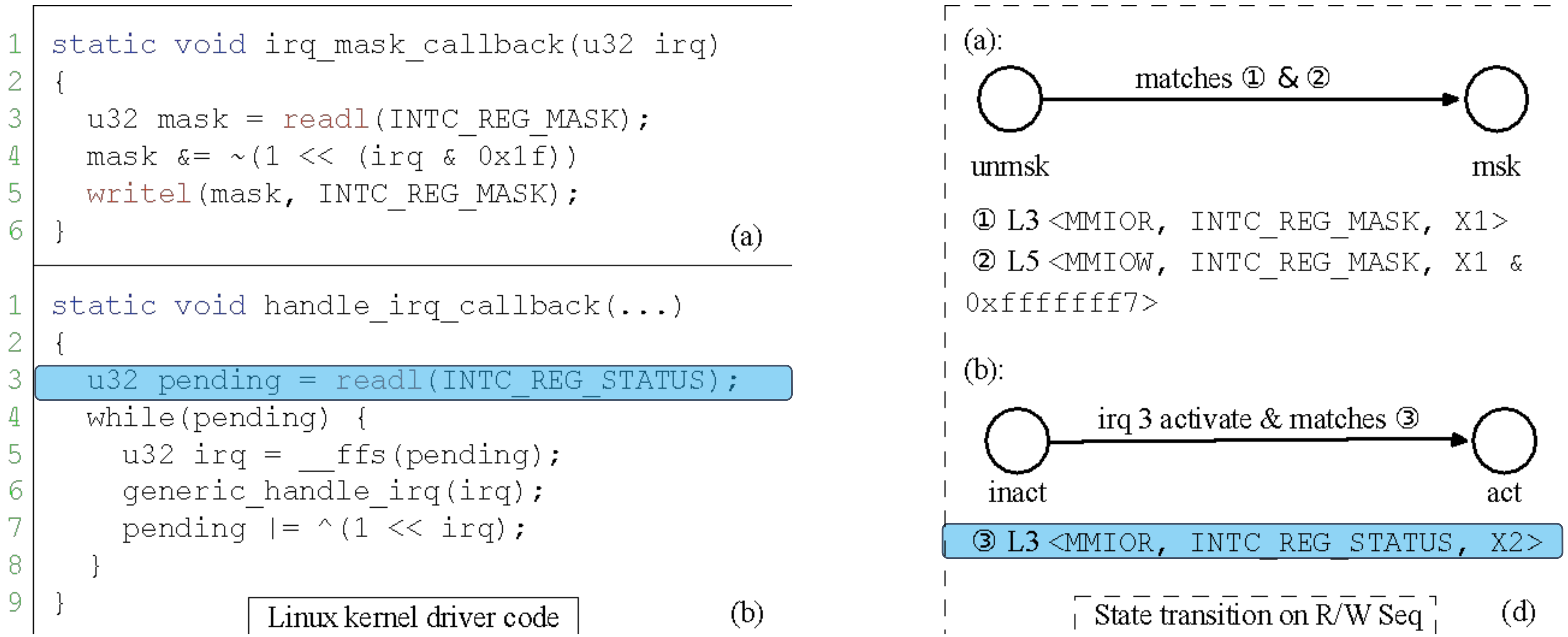
- **The MMIO Read/Write sequence from Linux kernel can be recognized to drive the state machine of our emulated peripherals**

Model-guided Kernel Execution: Running Example



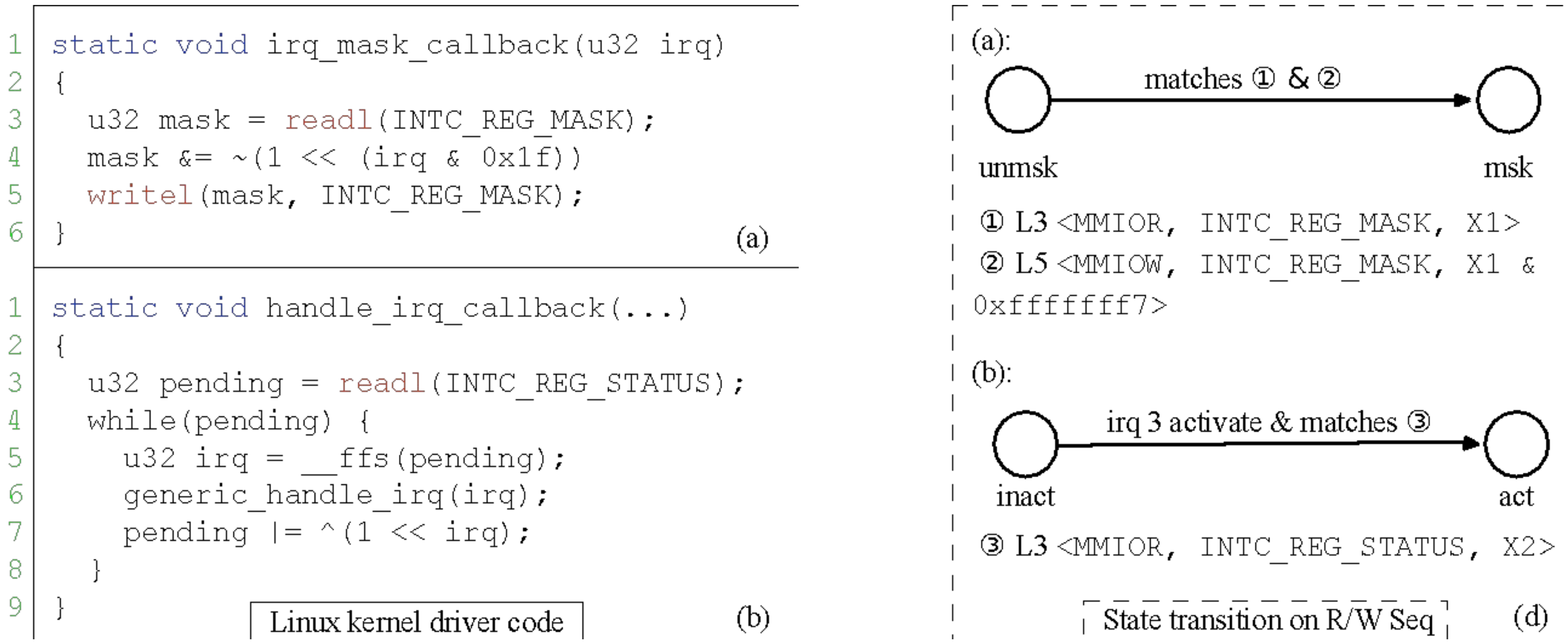
- **The MMIO Read/Write sequence from Linux kernel can be recognized to drive the state machine of our emulated peripherals**

Model-guided Kernel Execution: Running Example



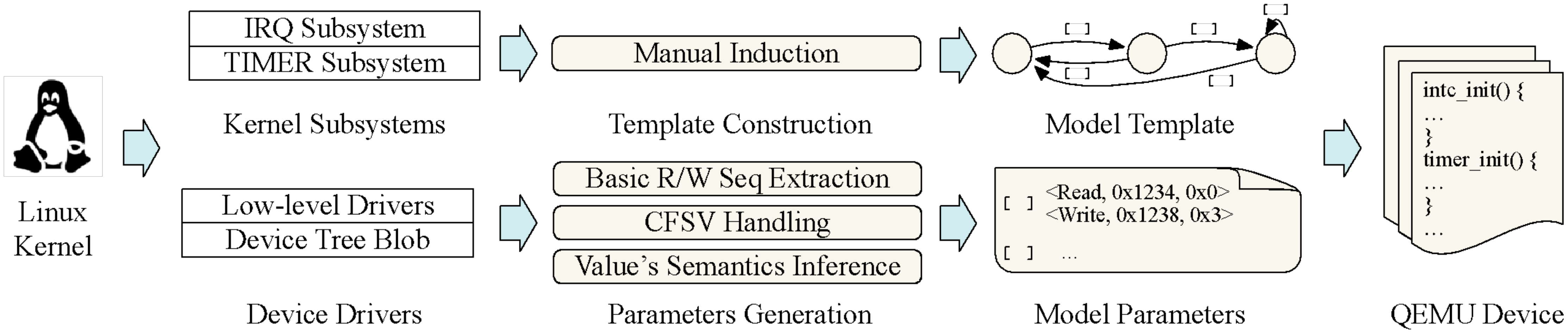
- **The MMIO Read/Write sequence from Linux kernel can be recognized to drive the state machine of our emulated peripherals**

Model-guided Kernel Execution: Running Example



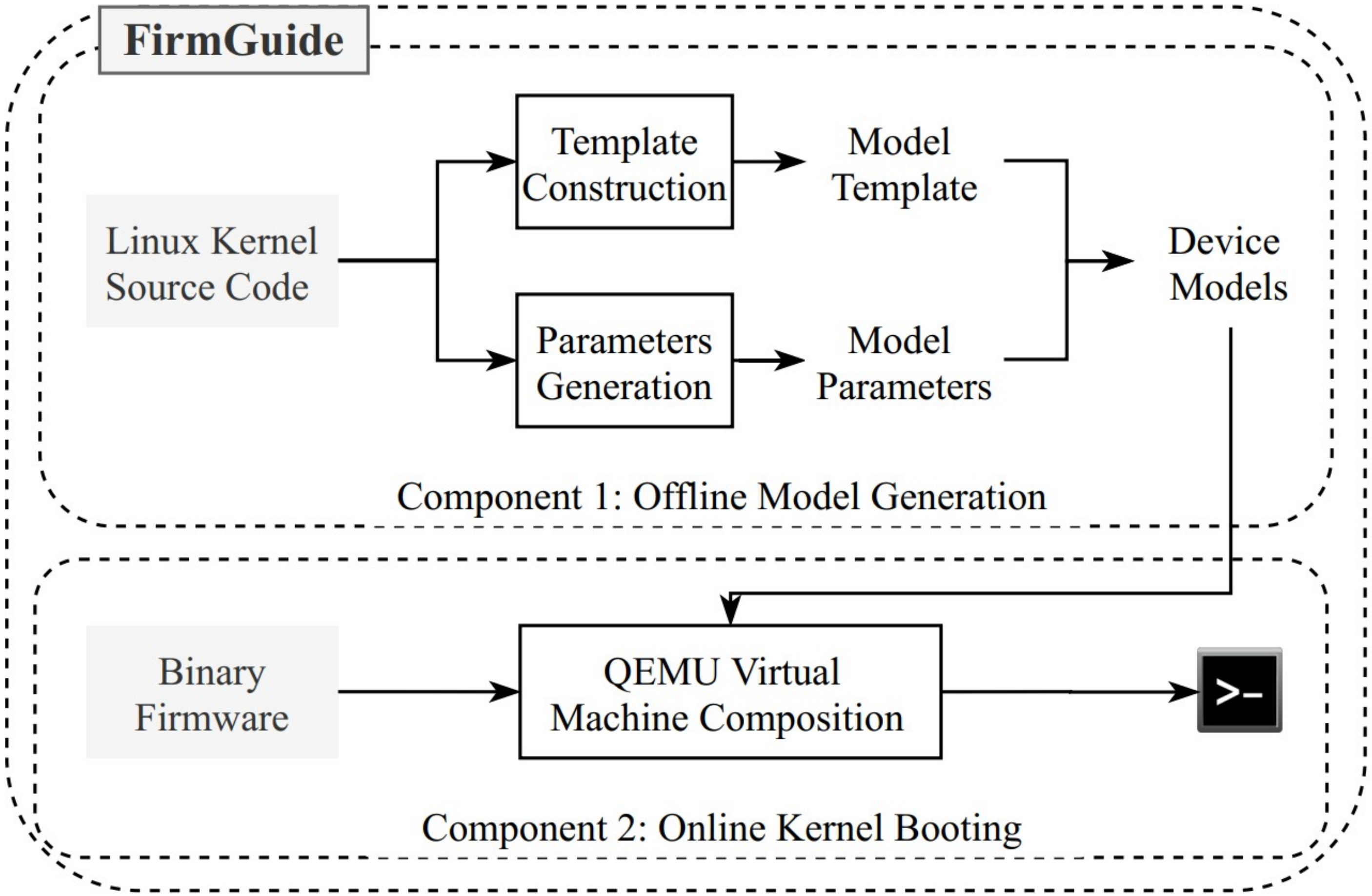
- **The MMIO Read/Write sequence from Linux kernel can be recognized to drive the state machine of our emulated peripherals**

Model-guided Kernel Execution: Methodology



- **We semi-automatically build the state machine of each peripheral: a general model template (manually) plus model parameters (automatically)**

System Design and Implementation



LLVM pass for preprocess
KLEE for MMIO R/W Seq
Python for glues

Python for main logic
Template-render pattern

Evaluation

RQ 1: What peripheral models can we generate?

Type I

Subtarget	Interrupt Controller	Timer	# of Paths	# of Solutions	First/All Solution (s)	Exists CFSV (y/n)	Timer Semantics	LoC
ramips/rt305x	ralink-rt2880-intc	`not necessary`	262	4	1/2	n	-	3,366
ath79/generic	qca,ar7240-intc	`not necessary`	110,083	1,134	5/943	n	-	4,138
kirkwood/generic	marvell,orion-intc marvell,orion-bridge-intc	marvell,orion-timer	132	2	2/3	y	$y =\sim x$	4,790
bcm53xx/generic	arm,cortex-a9-gic	arm,cortex-a9-global-timer arm,cortex-a9-twd-timer	150,336	2,592	2,027/24,070	y	$y = x_1 << 32 + x_2$	3,537
oxnas/generic	arm,arm11mp-gic	arm,arm11mp-twd-timer plxtech,nas782x-rps-timer	52,332	1,246	914/16,184	y	$y = x$	3,366

Type II

Subtarget	ramips/ rt305x	ath79/ generic	kirkwood/ generic	bcm53xx/ generic	oxnas/ generic
count	1/10	2/15	3/26	2/4	2/9

Evaluation

RQ 2: What embedded Linux kernels can we rehost?

SoC	Unpack	Kernel	Booting Validation	
			User Space	Shell
Ralink RT3050	1164	1164	1144 (98.28%)	1052 (90.38%)
Ralink RT3052	1815	1815	1815 (100.00%)	1661 (91.52%)
Ralink RT3352	173	173	173 (100.00%)	157 (90.75%)
Ralink RT5350	1632	1632	1611 (98.71%)	1475 (90.38%)
subtarget: ramips/rt305x	4784	4784	4743 (99.14%)	4345 (90.82%)
Atheros AR7161	36	36	20 (55.56%)	20 (55.56%)
Atheros AR7241	20	20	12 (60.00%)	12 (60.00%)
Atheros AR7242	24	24	24 (100.00%)	24 (100.00%)
Atheros AR9330	4	4	4 (100.00%)	4 (100.00%)
Atheros AR9331	24	24	12 (50.00%)	12 (50.00%)
Atheros AR9341	10	10	4 (40.00%)	4 (40.00%)
Atheros AR9342	24	24	24 100.00%)	24 (100.00%)
Atheros AR9344	70	70	64 (91.43%)	64 (91.43%)
Qualcomm Atheros QCA9531	22	22	16 (72.73%)	16 (72.73%)
Qualcomm Atheros QCA9533	41	41	14 (34.15%)	14 (34.15%)
Qualcomm Atheros QCA9557	64	64	64 (100.00%)	64 (100.00%)
Qualcomm Atheros QCA9558	54	54	50 (92.59%)	50 (92.59%)
Qualcomm Atheros QCA9560	16	16	16 (100.00%)	16 (100.00%)
Qualcomm Atheros QCA9561	18	18	14 (77.78%)	14 (77.78%)
Qualcomm Atheros QCA9563	114	114	106 (92.98%)	106 (92.98%)
subtarget: ath79/generic	541	541	444 (82.07%)	444 (82.07%)
Broadcom BCM4708A0	241	241	241 (100.00%)	241 (100.00%)
Broadcom BCM4709A0	128	128	128 (100.00%)	128 (100.00%)
Broadcom BCM47189	19	19	19 (100.00%)	19 (100.00%)
subtarget: bcm53xx/generic	388	388	388 (100%)	388 (100%)
Marvell 88F6192	20	20	20 (100.00%)	20 (100.00%)
Marvell 88F6281	208	204	204 (100.00%)	144 (70.59%)
Marvell 88F6282	102	102	100 (98.04%)	80 (78.43%)
subtarget: kirkwood/generic	330	326	324 (99.39%)	244 (74.85%) †
PLX NAS7820	149	149	48 (32.21%)	48 (32.21%)
subtarget: oxnas/generic	149	149	48 (32.21%)	48 (32.21%) ✦
Overall	6,192	6,188	5947 (96.11%)	5469 (88.38%)

Given 6K+ firmware crossing 10 vendors, 3 architectures, and 22 Linux kernel versions, FirmGuide can successfully rehost more than 96% of them.

Evaluation

RQ 3: What about the functionality of the rehosted embedded Linux kernels?

Linux Test Project: Syscall Testing

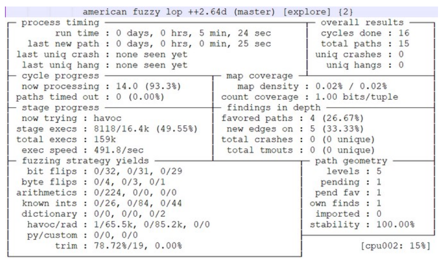
Models	Pass	Skipped	Failed	Total
Fully generated	1049	164	46	1259
Ground Truth	1049	164	46	1259

RQ 4: What are application of FirmGuide?

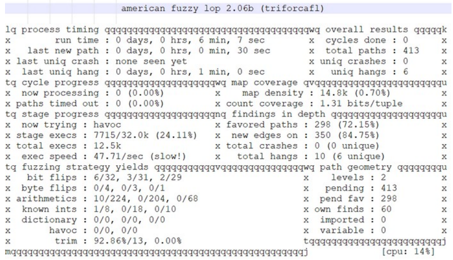
CVE Reproduction and Exploit Development

CVE ID	CVE Type	Status	Version
CVE-2016-5195	Race Condition	✗	N/A
CVE-2016-8655	Race Condition	✓ †	□
CVE-2016-9793	Integer Overflow	✓	□ ◇
CVE-2017-7038	Integer Overflow	✓ †	◇
CVE-2017-1000112	Buffer Overflow	✓ †	△
CVE-2018-5333	NULL Pointer Dereference	✓ †	□ ◇

Fuzzing



UnicoreFuzz



TriforceAF
L

Summary

Conclusion

- A novel technique “Model-Guided Kernel Execution” for peripheral modeling
- The first semi-automatic framework for embedded Linux kernel rehosting
- Feasible dynamically understanding and mining vulnerability in embedded kernels

Discussion

Limitation and future work

Manually state machine construction for more complex peripherals

High fidelity of Type-II peripherals

Q & A

qiangliu@zju.edu.cn, cen001@e.ntu.edu.sg