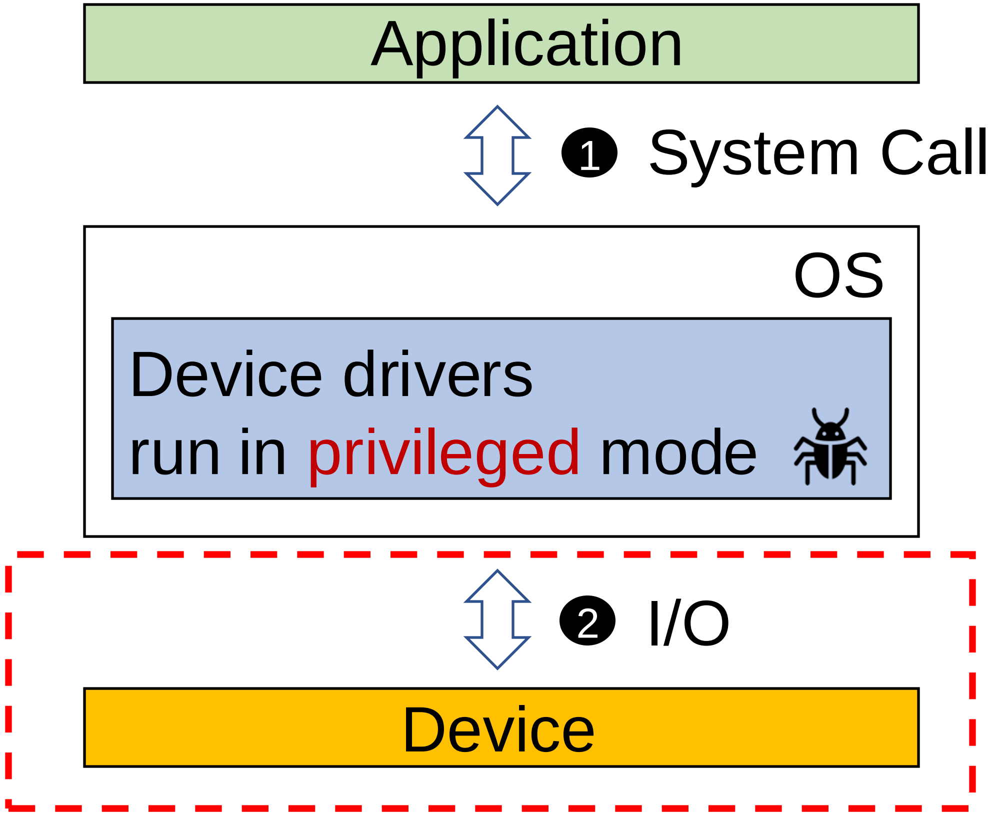
DEVFUZZ: Automatic Device Model-Guided Device Driver Fuzzing

Yilun Wu[*] Tong Zhang[+] Changhee Jung[‡] Dongyoon Lee[*]

[*] Stony Brook University [+] SAMSUNG [‡] PURDUE UNIVERSITY

Device Driver Security

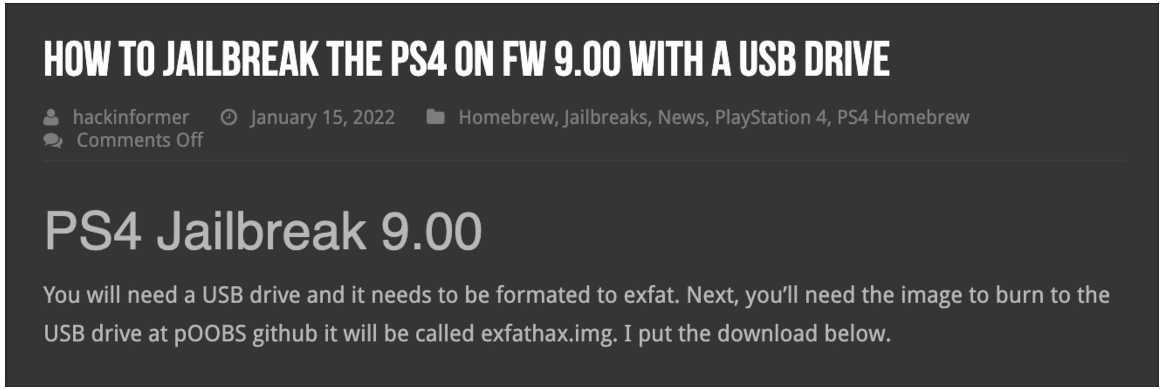
Two Interfaces



Threat Model

- An attacker can plug in a malicious device (e.g. USB hack stick)
- A device can feed malformed inputs to exploit security vulnerabilities in a device driver (e.g. buffer overflows)

Real World Examples

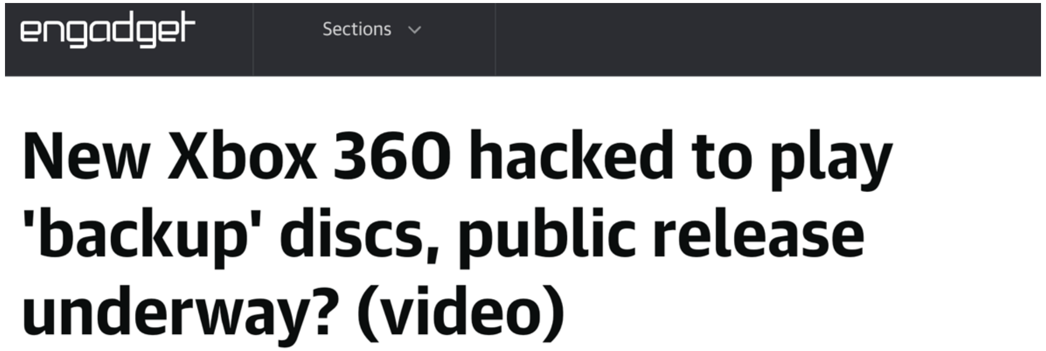


HOW TO JAILBREAK THE PS4 ON FW 9.00 WITH A USB DRIVE

hackinformer January 15, 2022 Homebrew, Jailbreaks, News, PlayStation 4, PS4 Homebrew
Comments Off

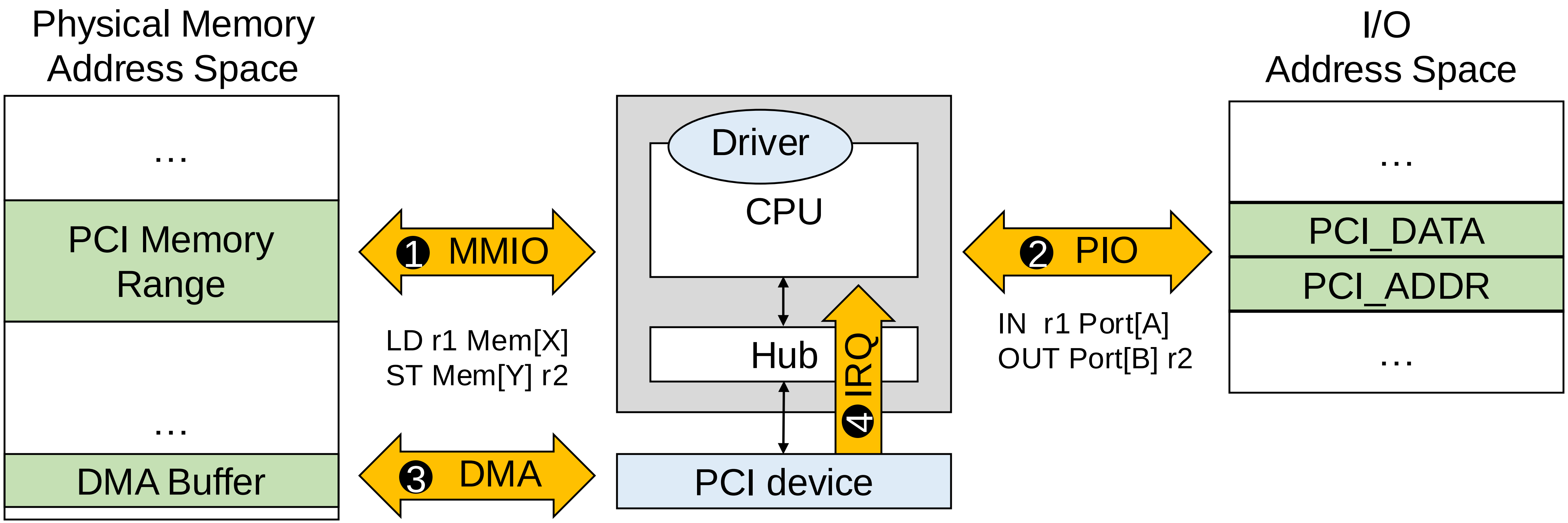
PS4 Jailbreak 9.00

You will need a USB drive and it needs to be formated to exfat. Next, you'll need the image to burn to the USB drive at pOOBS github it will be called exfathax.img. I put the download below.



New Xbox 360 hacked to play 'backup' discs, public release underway? (video)

Challenge 1: Large Device Input Space



❶ Memory Mapped IO (MMIO)

❷ Port IO (PIO)

❸ Direct Memory Access (DMA)

❹ Interrupt (IRQ)

Testing all possible input is unscalable and ineffective

Challenge 2: Dynamic Probing

- Many bus architectures (e.g., PCIe, USB) allow users to plug-in new devices.
- OS pairs a driver with a device and initialize it using a probing function.

```
int pcnet32_probe(struct pci_dev * pdev) {
    …
    void *ioaddr = pci_resource_start(pdev, 0);
    int err = -ENODEV;
    int chip_version;
    if (ioread(ioaddr+0x10) != 4 ||
        ioread(ioaddr+0x12) & 0xA) {
        return err;
    }
    chip_version = ioread(ioaddr+0x10) |
                   ioread(ioaddr+0x10) << 16);
    if (chip_version != 0xABCD) {
        return err;
    }
    …
    return 0;
}
```

pcnet32 network device driver probing function

Passing probing conditions require device-specific input

Can we test device drivers without actual devices?

Prior Work: Testing Device Drivers

Testing with real hardware

- e.g., PeriScope [NDSS'19]
- Hardware may not be readily available

Symbolic/Concolic execution

- e.g., SymDrive [OSDI'12], DriFuzz [SEC'22]
- Slow

Manual software model (for probing) + Fuzzing

- e.g., USBFuzz [SEC' 20]
- Unscalable. Error-prone

Static analysis (for probing) + Fuzzing

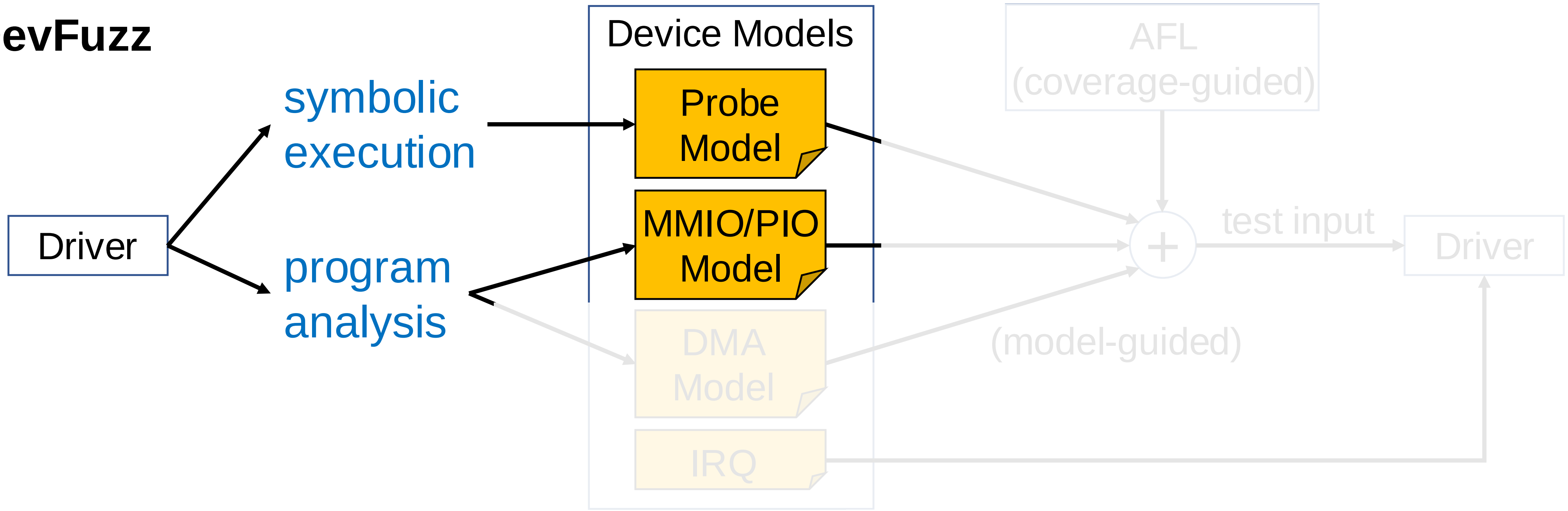
- e.g., PrIntFuzz [ISSTA'22]
- Low success rate for probing due to imprecise static analysis

Our Approach

Goals: Testing device drivers

- without actual devices
- without manual modeling
- without (input space) state explosion

DevFuzz



Step 1: automatic model generation

Step 2: model-guided fuzzing

Using Symbolic Execution for Probe Model

Built on S^{2}E [ASPLOS 2011]

- QEMU for emulation
- KLEE for symbolic execution

Symbolic Execution

- Run probing functions with symbolic MMIO/PIO address space regions
- Successful probing
  - Use the SMT solver to solve the constraint to get concretized values
- Failed probing
  - Terminate the case and explore alternative paths

“Concretized” Probe Model

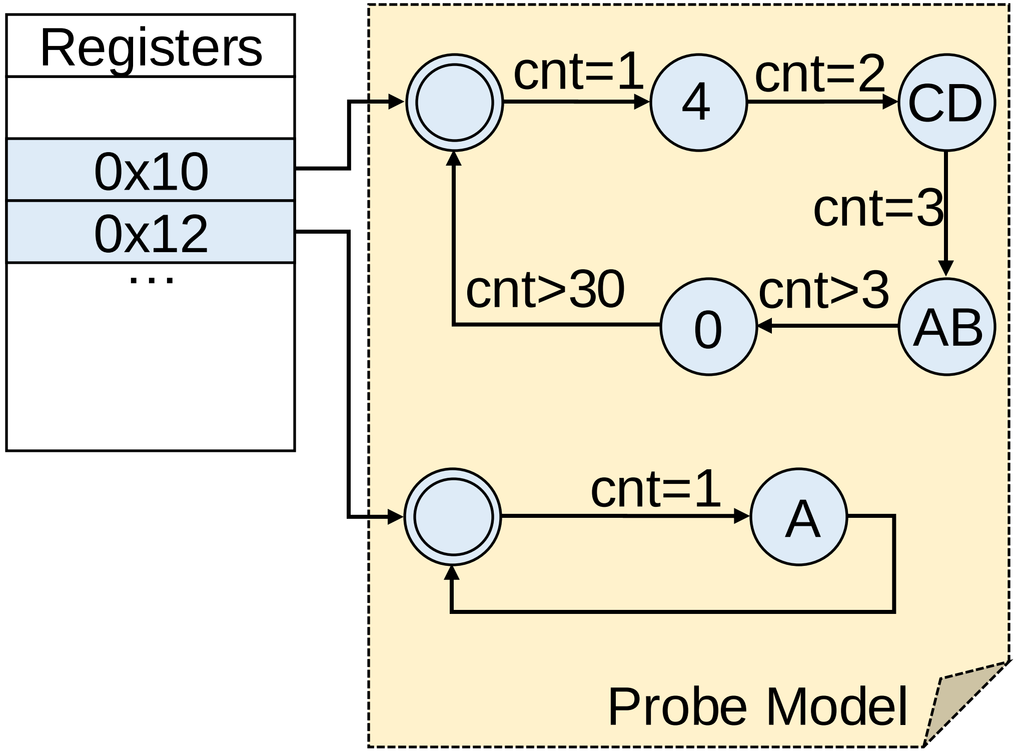
- Allow DevFuzz to pass (complex) probing path constraints

Probe Model Example

```c
int pcnet32_probe(struct pci_dev * pdev) {
    …
    void *ioaddr = pci_resource_start(pdev, 0);
    int err = -ENODEV;
    int chip_version;
    if (ioread(ioaddr+0x10) != 4 ||
        ioread(ioaddr+0x12) & 0xA) {
        return err;
    }
    chip_version = ioread(ioaddr+0x10) |
                   ioread(ioaddr+0x10) << 16);
    if (chip_version != 0xABCD) {
        return err;
    }
    …
    return 0;
}
```

pcnet32 network device driver probing function

MMIO Address Space



A state machine of device register values

Using Program Analysis for MMIO/PIO Models

LLVM-based Static Program Analysis

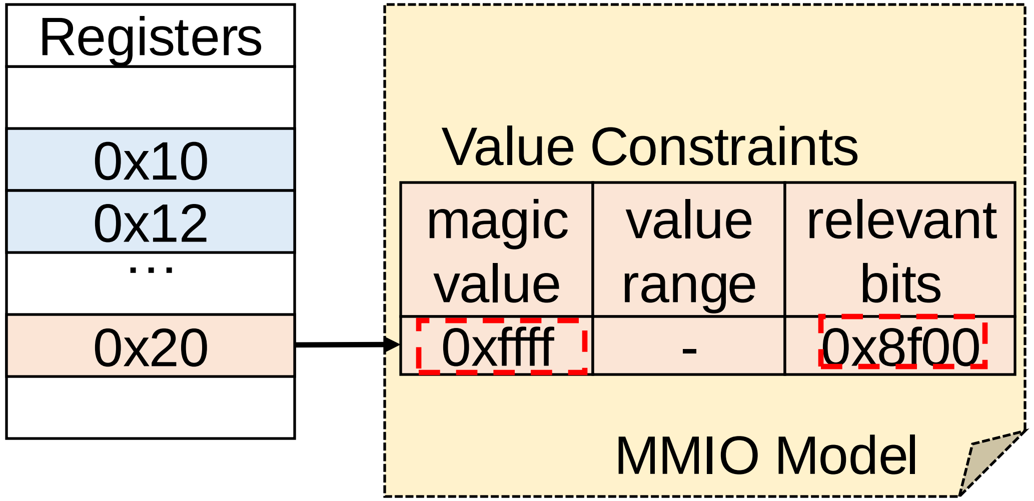
```
…
csr0 = lp->a->read_csr (ioaddr, CSR0);
while ((csr0 & 0x8f00) && --boguscnt >= 0)
{
    if (csr0 == 0xffff)
        break;
    lp->a->write_csr (ioaddr, CSR0,
                      csr0 & ~0x004f);
    if (csr0 == 0x4000) {
        …
    }
    if (csr0 == 0x1000) {
        …
    }
    csr0 = lp->a->read_csr (ioaddr, CSR0);
}
…
```

pcnet32 network device driver interrupt handler

❶ IO wrapper analysis

❷ IO address analysis

❸ IO value flow analysis



magic value	value range	relevant bits
0xffff	-	0x8f00

Guides fuzzing inputs

And More …

DMA Model

- DevFuzz uses dynamic/static program analyses
- DMA buffer address/shape analysis

IRQ

- Simple model
- Generate IRQs using a timer

Model Generality and Reusability

- The generated Probe, MMIO, PIO Models reflect device-specific properties
- The models generated from one OS (Linux) can be reused to test device drivers of another OS (FreeBSD or Windows)

Evaluation Summary

- Large-scale security evaluation
  - Tested 150 Linux drivers
  - Reused device models to test 25 FreeBSD and 16 Windows drivers
- Small-scale code coverage evaluation
  - 17 network device drivers
  - Compared with prior work: PrintFuzz [ISSTA'22] and DriFuzz [SEC'22]
  - Compared with manually-developed QEMU device models (not shown in this talk)

Security Evaluation

OS	Tested	Probed	Bugs/Crash	Patched
Linux	150	112	63	39
FreeBSD	25	14	8	2
Windows	16	8	1	0
All	191	134	72	41

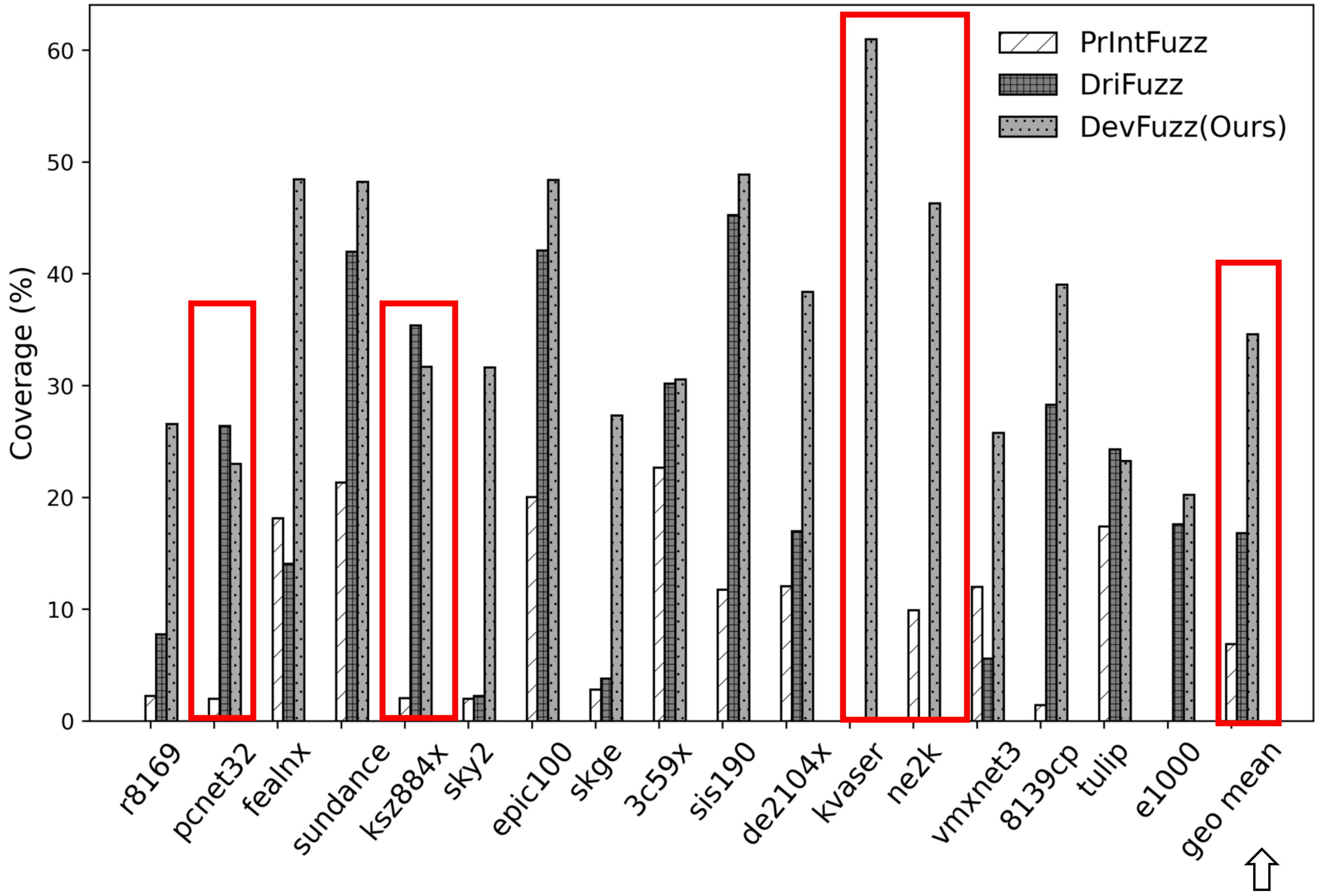
- For Linux: 75% (112/150) were successfully probed via symbolic execution
  - Some unsupported features (e.g., IRQ during symbolic execution)
  - Complex path constraints (e.g., checksum)
- For FreeBSD/Windows: About half Probe Models were reusable
- 72 Bugs (1 CVE) were reported (including FreeBSD/Windows cases)
- 56% (41/72) were patched to the mainstream

Coverage Comparison with Prior Works

- PrintFuzz [ISSTA'22] uses static analysis to pass probing path constraints, followed by fuzzing

- DriFuzz [SEC'22] uses concolic execution

- DevFuzz achieves better
    - Successful probing rate
    - Code coverage



Geometric Mean

Conclusion

- DevFuzz leverages symbolic execution, program analysis, and fuzzing to enable testing device drivers
  - without actual devices
  - without manual device modeling
  - without (input space) state explosion
- DevFuzz uncovered 72 bugs (41 patched)
- DevFuzz achieved higher code coverage than prior works
- DevFuzz were able to test a large set of device drivers without devices across three different OSes (Linux, FreeBSD, and Windows

Q&A