Towards Full-Lifecycle Security Enforcement of Hypervisors

Qiang Liu, PostDoc

About Qiang Liu

2018.09 - 2023.09: PhD@ZJU with Prof. Yajin Zhou

2023.11 - current: PostDoc@EPFL with Prof. Mathias Payer

Research Topics System Security

IoT/Cloud -> AI Systems

Vulnerabilities, Offensive Research -> Defensive Research

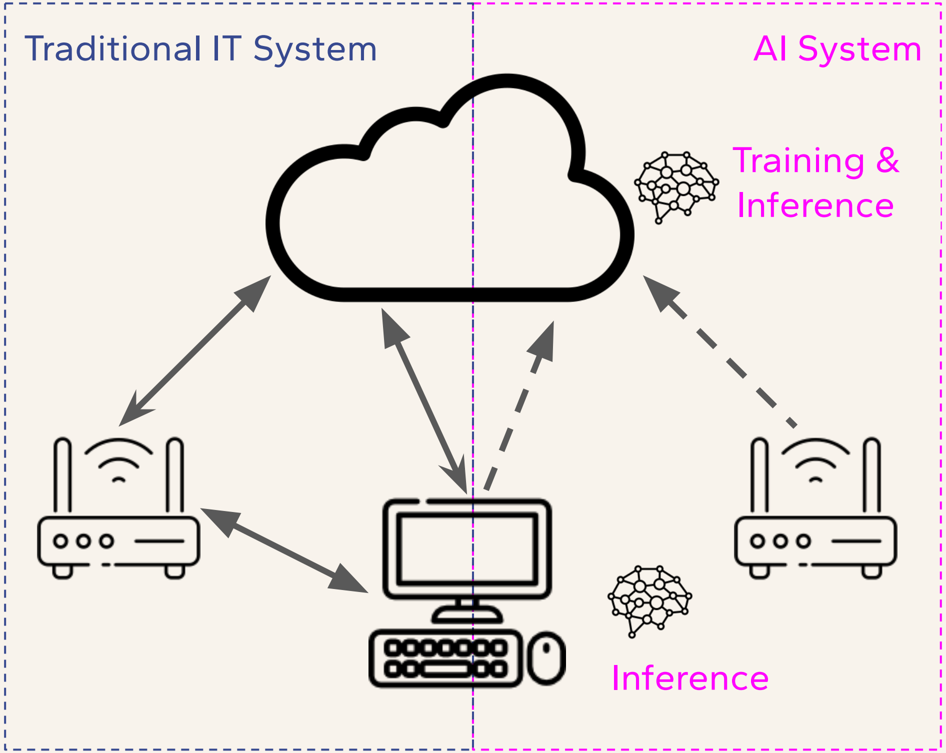
Secure Collaborative Computing/Chain of Trust

Browser Security & Interpreter Security
Reflector [AsiaCCS25]

Network Protocol Security
Tango [RAID24] 🏆
BGPFuzz [WIP]

Hypervisor Security
HyperPill [SEC24]🏆
ViDeZZo [S&P23]
Truman [NDSS25]

Firmware Rehosting
FirmGuide [ASE21]
ECMO [CCS21]



RQ1: How to protect training and deployment from security threats?

RQ2: How to prevent model misuse and achieve confidential inference?

RQ3: How to ensure the integrity and confidentiality of large models?

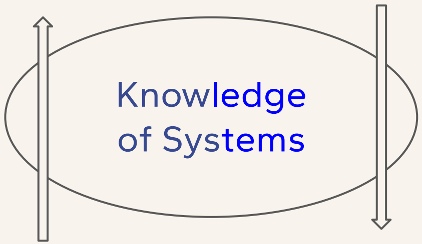
A Full-Lifecycle Enforcement of System Security

Ahead-of-release bug fixes

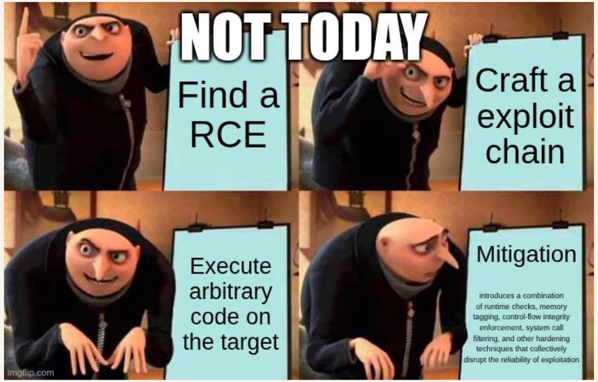
In-production attack mitigation

Source

Binary



Exploit



Exploitation as motivation/evaluation

Outline

Introduction to Hypervisors

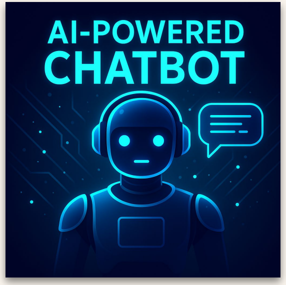
Hypervisors: Ahead-of-Release Bug Fixes

Hypervisors: In-Production Attack Mitigation

Open Questions, Future Work, and Conclusion

A Predestined Journey to the Cloud

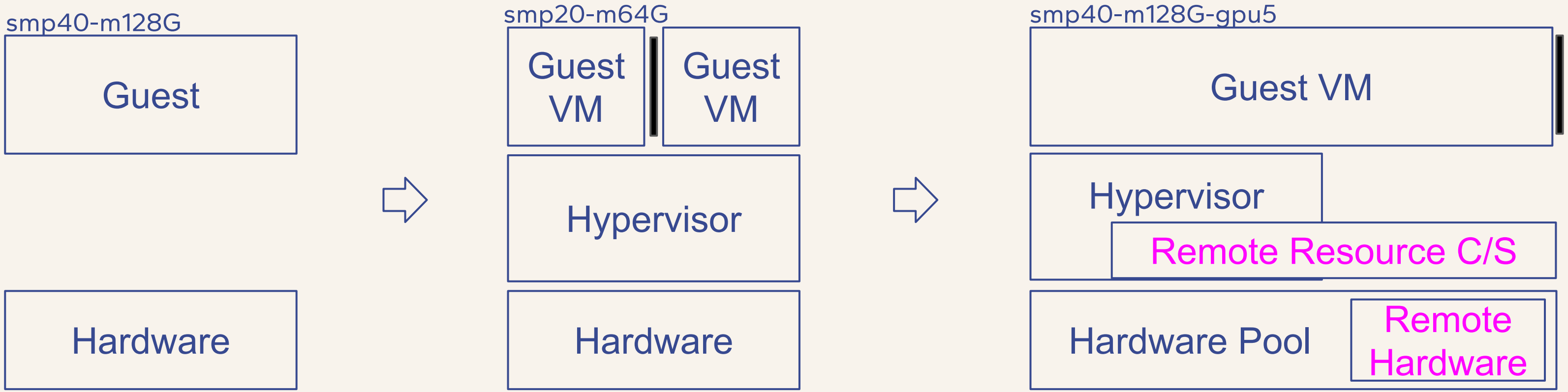
A friend of mine is building AI-powered services locally, ...



With Cloud computing, AI services can be both scalable and efficient. This is made possible by the hypervisor.

Hypervisors, Virtual Machines, and Isolation

No Hypervisor | **With Hypervisor** | **With Hardware Pool**



A guest accesses hardware directly

A guest appears to control hardware upon request with isolation

A guest scales out resources like GPUs on demand for AI workloads with isolation

Choose Your Favorite Hypervisor

I'm closed-source and Type-I (baremental)

I'm closed-source and love Microsoft

I'm an open-source superman; nothing can beat me

I'm written in Rust

Hypervisor.Framework - Virtualization on macOS

I'm a macOS native hypervisor and support both x86_64 and arm64

I'm Xen — old but gold, and still refusing to retire

I'm also written in Rust and slim

Hypervisors Everywhere

Business & Infrastructure

- Cloud computing platforms, smartphones, smart vehicles, base station units (e.g., 5G/4G towers), routers and gateways, industrial control systems

Security Applications

- Malware analysis, Honeypots, Intrusion detection, Confidential computing
- An alternative to the kernel as the trusted computing base (TCB)

Rehosting is the process of migrating firmware to a virtualized execution environment. We contributed Linux kernel–based rehosting solutions in FirmGuide [ASE'21] and ECMO [CCS'21].

Attacker’s Gain

- VM escape
- Data exfiltration
- Privilege escalation
- Service disruption / DoS
- Stealth persistence
- Horizontal move

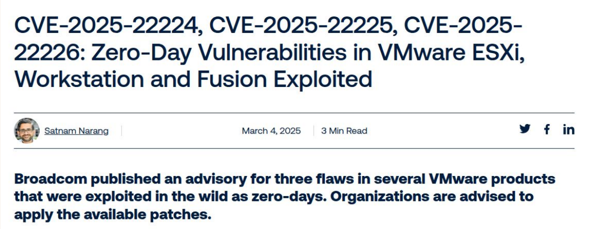
Microsoft fixes under-attack privilege-escalation holes in Hyper-V

Plus: Excel hell, angst for Adobe fans, and life's too Snort for Cisco

Iain Thomson   Wed 15 Jan 2025 // 01:33 UTC

PATCH TUESDAY The first Patch Tuesday of 2025 has seen Microsoft address three under-attack privilege-escalation flaws in its Hyper-V hypervisor, plus plenty more problems that deserve your attention.

The Hyper-V vulnerabilities are CVE-2025-21333, CVE-2025-21334, and CVE-2025-21335, and were already being exploited in the wild as zero-days. They are rated important in terms of se



CVE-2025-22224, CVE-2025-22225, CVE-2025-22226: Zero-Day Vulnerabilities in VMware ESXi, Workstation and Fusion Exploited

Satnam Narang | March 4, 2025 | 3 Min Read

Broadcom published an advisory for three flaws in several VMware products that were exploited in the wild as zero-days. Organizations are advised to apply the available patches.

Attacker’s Gain

- VM escape
- Data exfiltration
- Privilege escalation
- Service disruption / DoS
- Stealth persistence
- Horizontal move

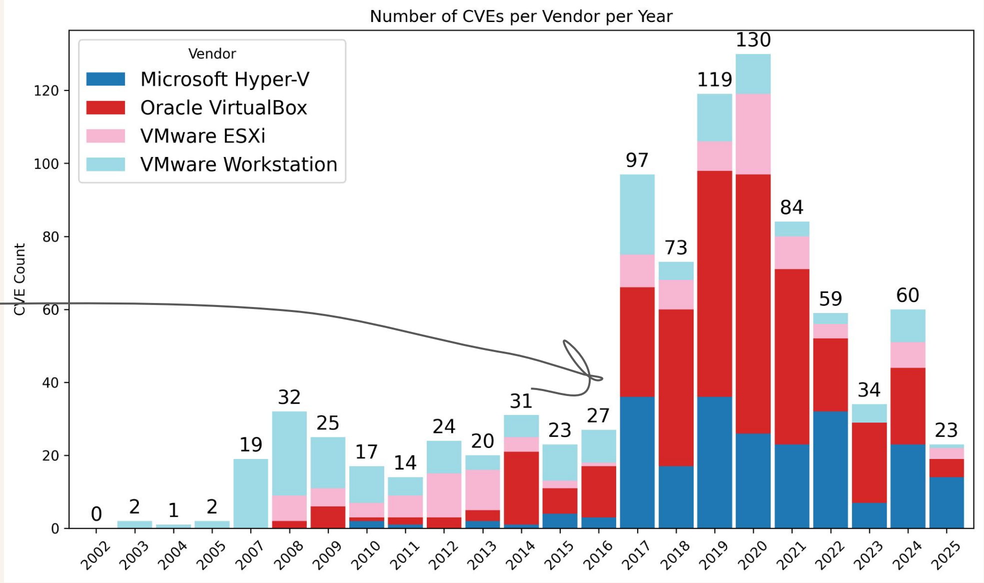
First time introduced in 2016

Pwn2Own’25 Virtualization Category	
Oracle VirtualBox	$40K
VMware Workstation	$80K
VMware ESXi	$150K
Microsoft Hyper-V	$250K


VULNERABILITIES

VENOM Vulnerability Opens Millions of Virtual Machines to Attack

May 18, 2015 by Pierluigi Paganini



QEMU Fuzzing

- 2015: VENOM VM Escape
- 2015-2017: Fuzzing initiatives
  - 360 Marvel Team/Micro Trend
- 2019: QEMU Fuzzing@GSC'19
  - Added to OSS-Fuzz in 2020
- 2021: QEMU Security Requirements
  - Raising the bar to assign CVEs
- 2021-2025: We reported ~70 security bugs to QEMU

QEMU security has been improved a lot

Attacker's Cost

Cost ↘

- Fuzzing tools
- More bug reports
- AI for cybersecurity

Now shifting to Linux/KVM and closed-source

- HyperPill
- First tool to analyze arbitrary x86/AArch64 and open-source/closed-source hypervisors across all major attack-surfaces (i.e., PIO/MMIO/Hypercalls/DMA)
- Discord: https://discord.gg/dxdvHvrK8D
- More human/funding resources requested to commercialize it

Outline

Introduction to Hypervisors

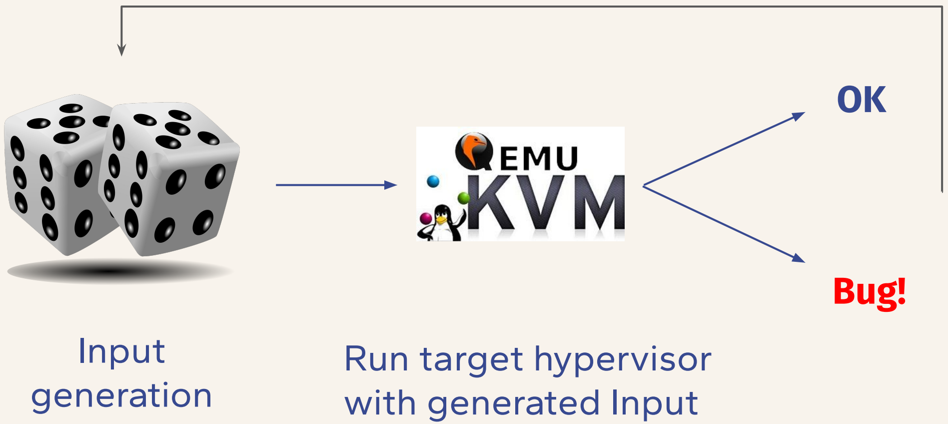
Hypervisors: Ahead-of-Release Bug Fixes

Hypervisors: In-Production Attack Mitigation

Open Questions, Future Work, and Conclusion

Hypervisors: Ahead-of-Release Bug Fixes

- **Fuzzing:** scalable to large code size and effective for bug discovery
- **Threat model**: the guest VM is not trusted; the attacker has the root privilege



Hypervisors: Ahead-of-Release Bug Fixes

- **Fuzzing:** scalable to large code size and effective for bug discovery
- **Threat model**: the guest VM is not trusted; the attacker has the root privilege

	Research Question	Solution	Key Results
Execution Environment	How to drive arbitrary hypervisors in a unified framework?	A snapshot-based Hypervisor Dock (HyperPill [SEC24]🏆)	First tool to analyze arbitrary x86/AArch64 and open-source/closed-source hypervisors across all major attack-surfaces (i.e., PIO/MMIO/Hypercalls/DMA)
Input Generation	How to generate high-quality inputs for hypervisor testing?	Dependency-Aware Input Generation (ViDeZZo [SP23] Truman [NDSS25])	Three kinds of dependencies for 29 virtual devices (including virtio), covering five categories, i.e., audio, storage, network, display, and USB

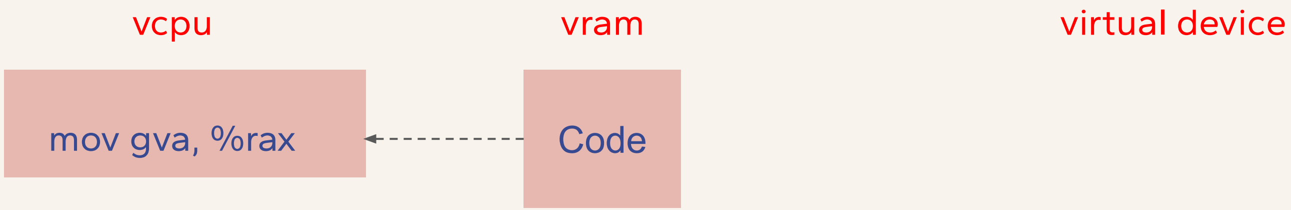
Execute, Translate, and Trap-Emulate-Return

vcpu

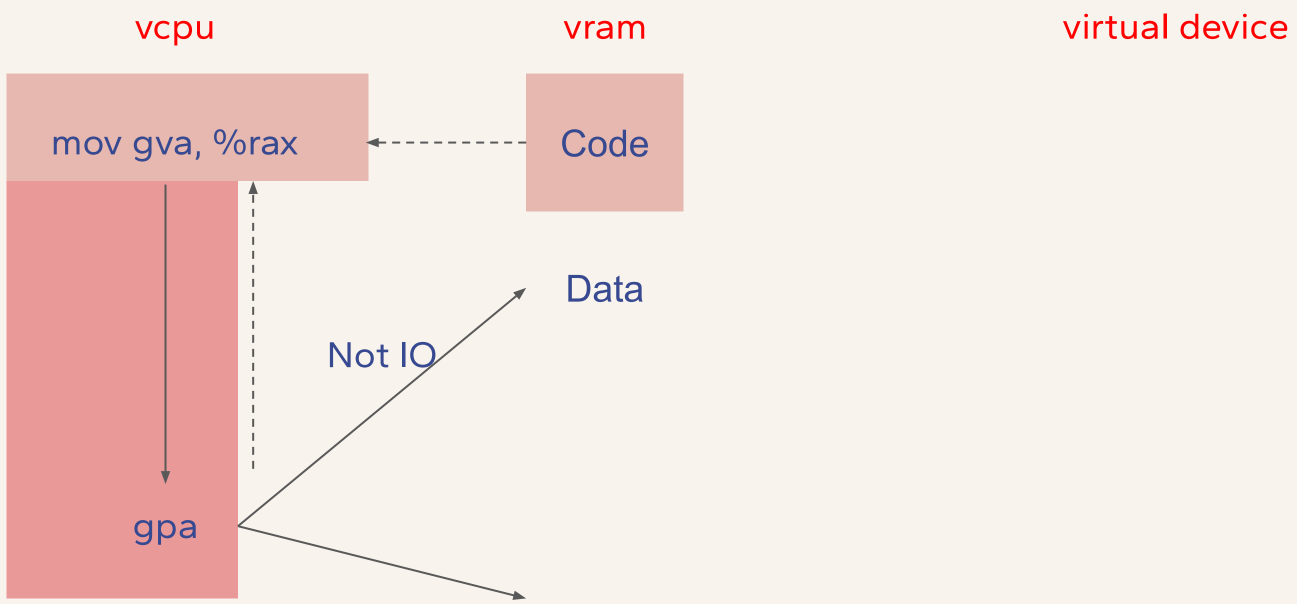
vram

virtual device

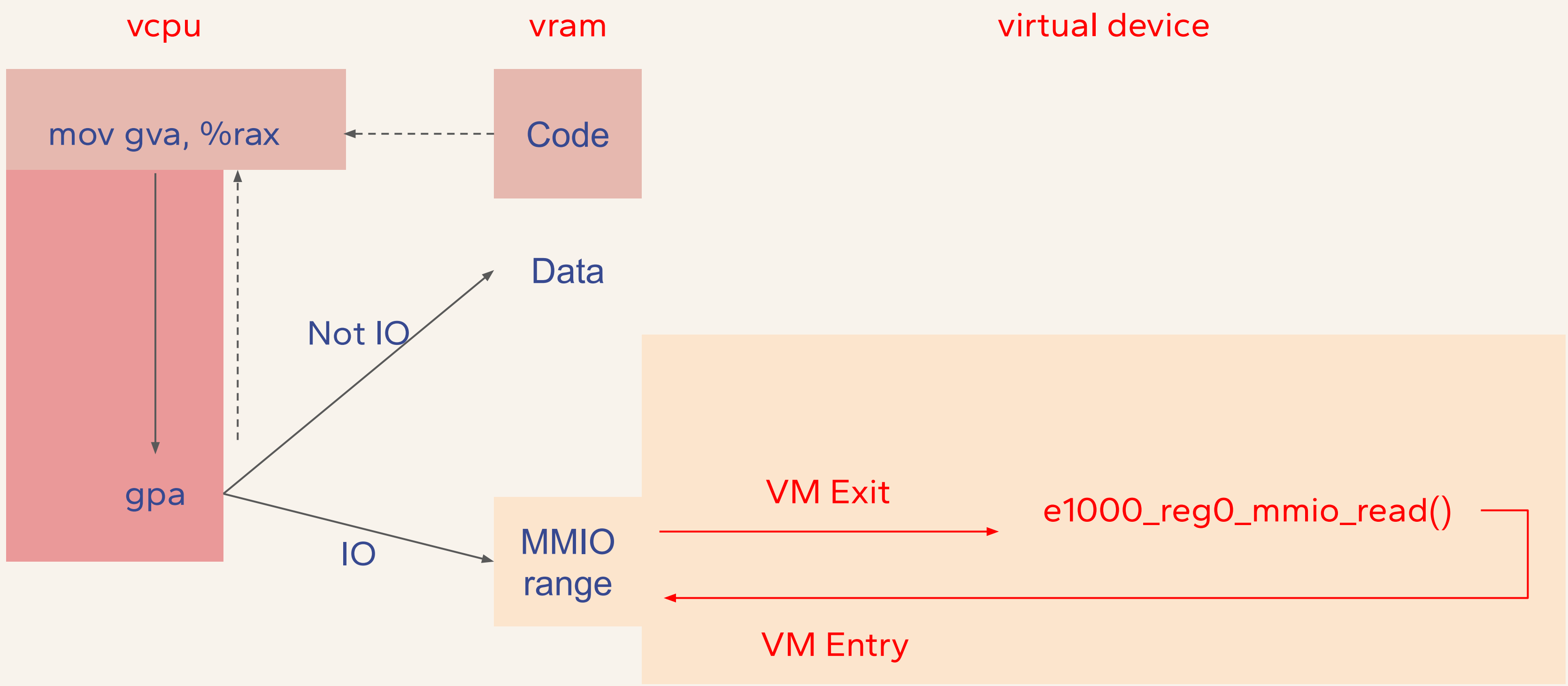
Execute, Translate, and Trap-Emulate-Return



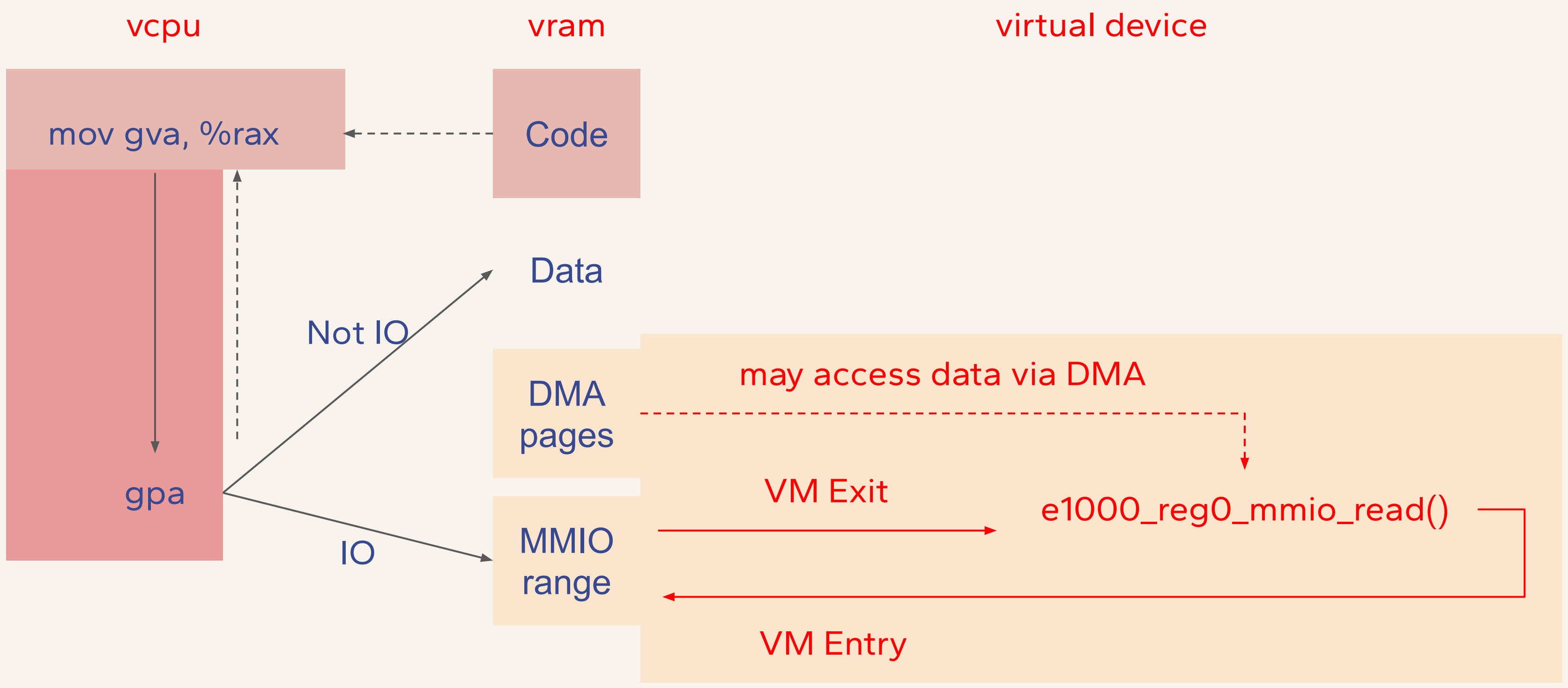
Execute, Translate, and Trap-Emulate-Return



Execute, Translate, and Trap-Emulate-Return



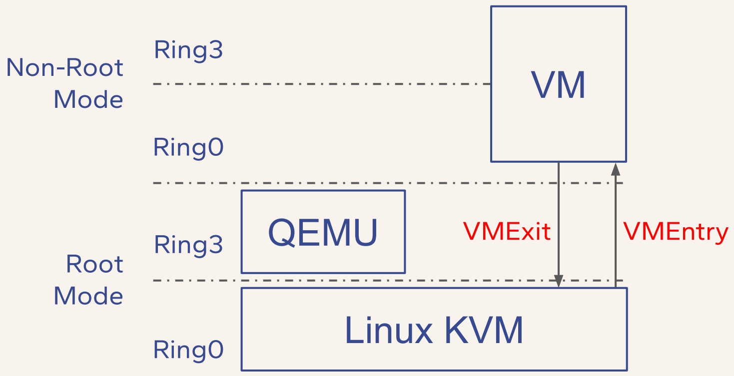
Execute, Translate, and Trap-Emulate-Return



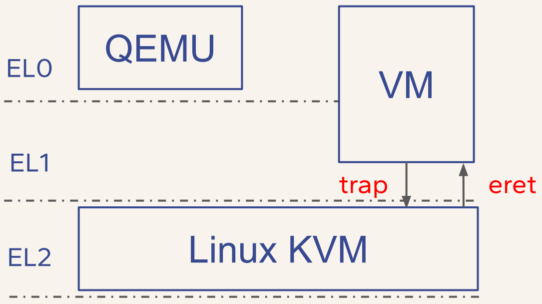
A snapshot-based Hypervisor Dock HyperPill [SEC24]🏆

What do all the hypervisors have in common?

- Trap-Emulate-Return: execute most guest instructions natively on hardware but trap and emulate "certain" instructions, allowing us to have a unified view of hypervisors



x86 virtualization

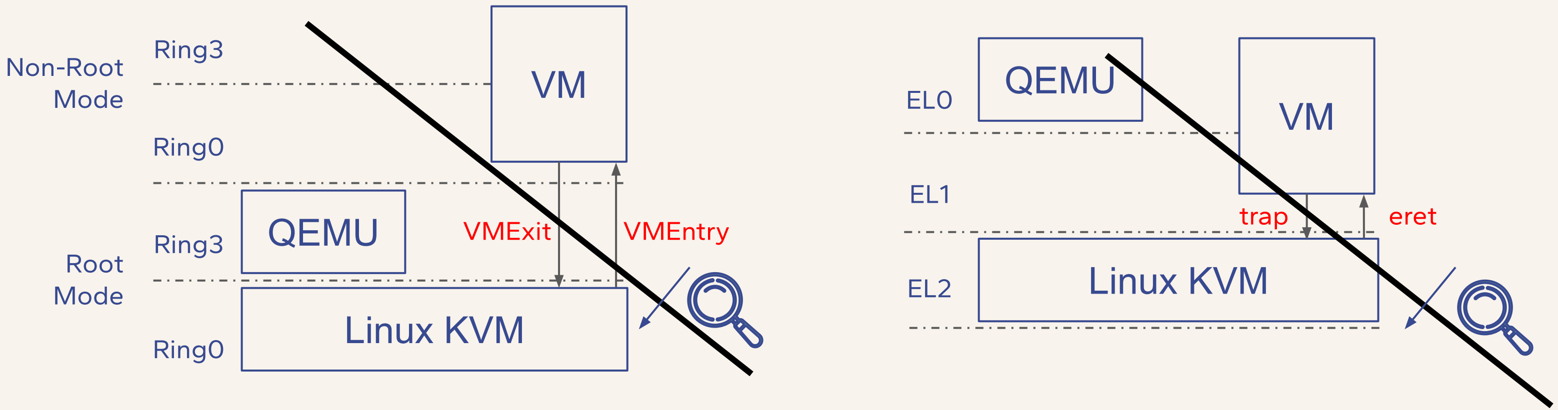


ARM64 virtualization

A snapshot-based Hypervisor Dock HyperPill [SEC24]🏆

What do all the hypervisors have in common?

- Trap-Emulate-Return: execute most guest instructions natively on hardware but trap and emulate "certain" instructions, allowing us to have a unified view of hypervisors



x86 virtualization

ARM64 virtualization

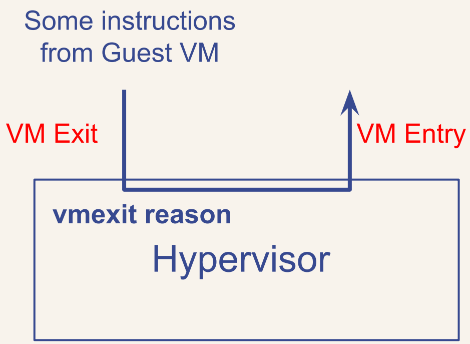
A snapshot-based Hypervisor Dock HyperPill [SEC24]🏆

A unified view of hypervisors

- Trap (vmexit reason)
- Emulate (may access DMA pages)
- Return (can be captured)

Hypervisor: VM Exit driven, Iterative program

- 😃 Perfect fuzzing target



A unified view of hypervisors

A snapshot-based Hypervisor Dock HyperPill [SEC24]🏆

A unified view of hypervisors

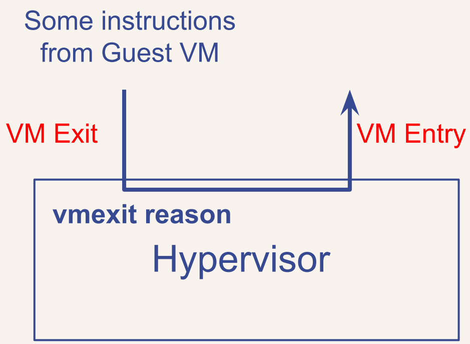
- Trap (vmexit reason)
- Emulate (may access DMA pages)
- Return (can be captured)

Hypervisor: VM Exit driven, Iterative program

- 😃 Perfect fuzzing target

Snapshot of the system status enables fine-grained control

- Hypervisor code and data (vmexit reason)
- Guest memory for DMA

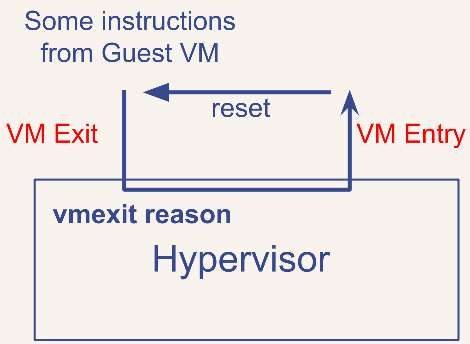


A unified view of hypervisors

A snapshot-based Hypervisor Dock HyperPill [SEC24]🏆

Four steps to drive a hypervisor to execute a sequence of VM exits

1. Modify the vmexit reason and its parameters
   - VMCS (x86), ESR/FAR/HPFAR_EL2 (ARM)
2. Run the hypervisor to process this VM exit
   - Provide DMA data on demand
   - VM Entry: vmresume (x86), eret to EL1 (ARM)
3. Partially reset the snapshot and issue a next VM exit
4. Fully reset the snapshot
   - All system registers, dirty pages

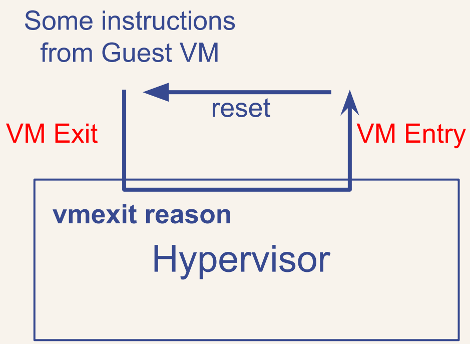


A unified view of hypervisors

Dependency-Aware Input Generation ViDeZZo [SP23] Truman [NDSS25]

A subset of VM messages that a hypervisor can take

1. Port I/O (PIO)
   - in/out (x86 only)
2. Memory-Mapped I/O (MMIO)
   - mov (x86), ld/st (ARM)
3. Prefilled memory for DMA requests (no trap)
   - mov (x86), ld/st (ARM)

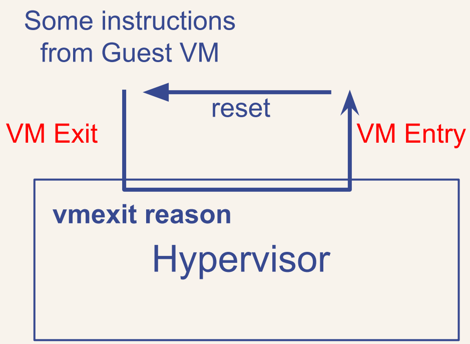


A unified view of hypervisors

Dependency-Aware Input Generation ViDeZZo [SP23] Truman [NDSS25]

A typical sequence of VM messages

- io_write()*rand()
- mem_write_for_dma()*rand()
- io_write()*1

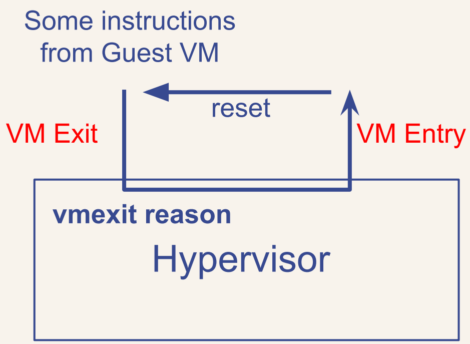


A unified view of hypervisors

Dependency-Aware Input Generation ViDeZZo [SP23] Truman [NDSS25]

A typical sequence of VM messages

- io_write()*rand() -> crash
- mem_write_for_dma()*rand()
- io_write()*1 -> crash

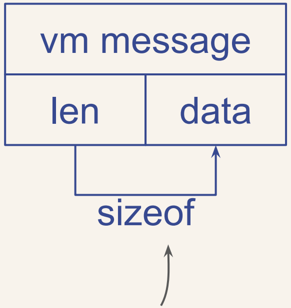


A unified view of hypervisors

Dependency-Aware Input Generation ViDeZZo [SP23] Truman [NDSS25]

A typical sequence of VM messages

- io_write()*rand()
- mem_write_for_dma()*rand()
- io_write()*1



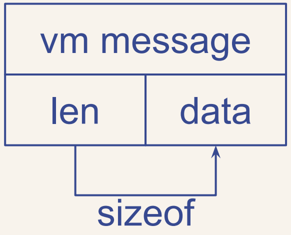
Three dependencies

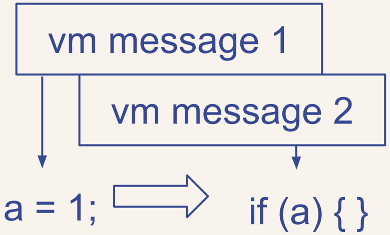
- Intra-message dependency: A field in a message may be dependent on another field

Dependency-Aware Input Generation ViDeZZo [SP23] Truman [NDSS25]

A typical sequence of VM messages

- io_write()*rand()
- mem_write_for_dma()*rand()
- io_write()*1





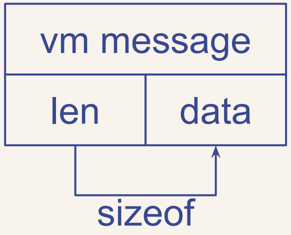
Three dependencies

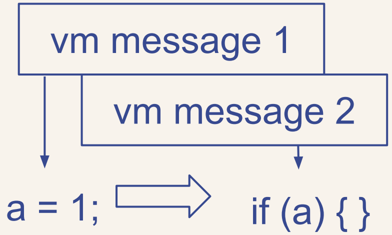
- Intra-message dependency: A field in a message may be dependent on another field
- Inter-message dependency: A message may depend on a previously issued message

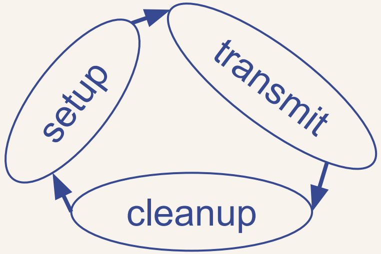
Dependency-Aware Input Generation ViDeZZo [SP23] Truman [NDSS25]

A typical sequence of VM messages

- io_write()*rand()
- mem_write_for_dma()*rand()
- io_write()*1







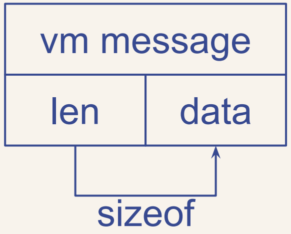
Three dependencies

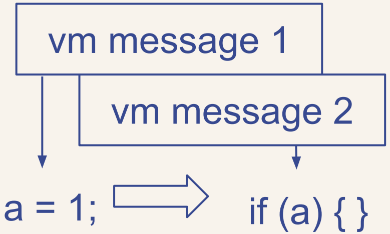
- Intra-message dependency: A field in a message may be dependent on another field
- Inter-message dependency: A message may depend on a previously issued message
- State dependency: A (bus-hidden) component follows a finite state machine

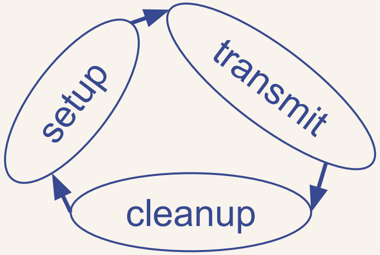
Dependency-Aware Input Generation ViDeZZo [SP23] Truman [NDSS25]

A typical sequence of VM messages

- io_write()*rand()
- mem_write_for_dma()*rand()
- io_write()*1





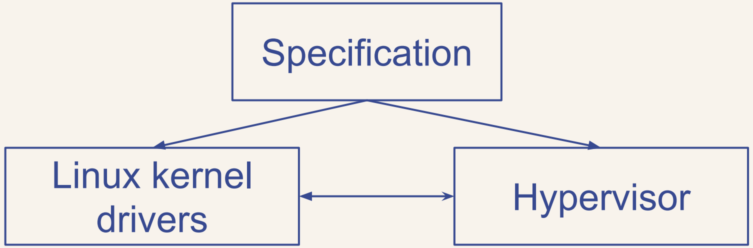


Three dependencies

- Intra-message dependency: A field in a message may be dependent on another field
- Inter-message dependency: A message may depend on a previously issued message
- State dependency: A (bus-hidden) component follows a finite state machine

Automatic extraction of three dependencies

- Knowledge is encoded in different formats
  - From hypervisor code, hard
  - From the Linux kernel drivers, easier



Hypervisors: Ahead-of-Release Bug Fixes

- **Fuzzing:** scalable to large code size and effective for bug discovery
- **Threat model**: the guest VM is not trusted; the attacker has the root privilege
- **Limitations**: KVM not covered; lacking of sanitizers for closed-source hypervisors

	Research Question	Solution	Key Results
Execution Environment	How to drive arbitrary hypervisors in a unified framework?	A snapshot-based Hypervisor Dock (HyperPill [SEC24]🏆)	First tool to analyze arbitrary x86/AArch64 and open-source/closed-source hypervisors across all major attack-surfaces (i.e., PIO/MMIO/Hypercalls/DMA)
Input Generation	How to generate high-quality inputs for hypervisor testing?	Knowledge-based Input Generation (ViDeZZo [SP23] Truman [NDSS25])	Three kinds of dependencies for 29 virtual devices (including virtio), covering five categories, i.e., audio, storage, network, display, and USB

Outline

Hypervisors: In-Production Attack Mitigation

De-privileging

Adapt existing hypervisor code to enforce the principle of least privilege

Formal Verification

Adapt an existing hypervisor for verification against security properties

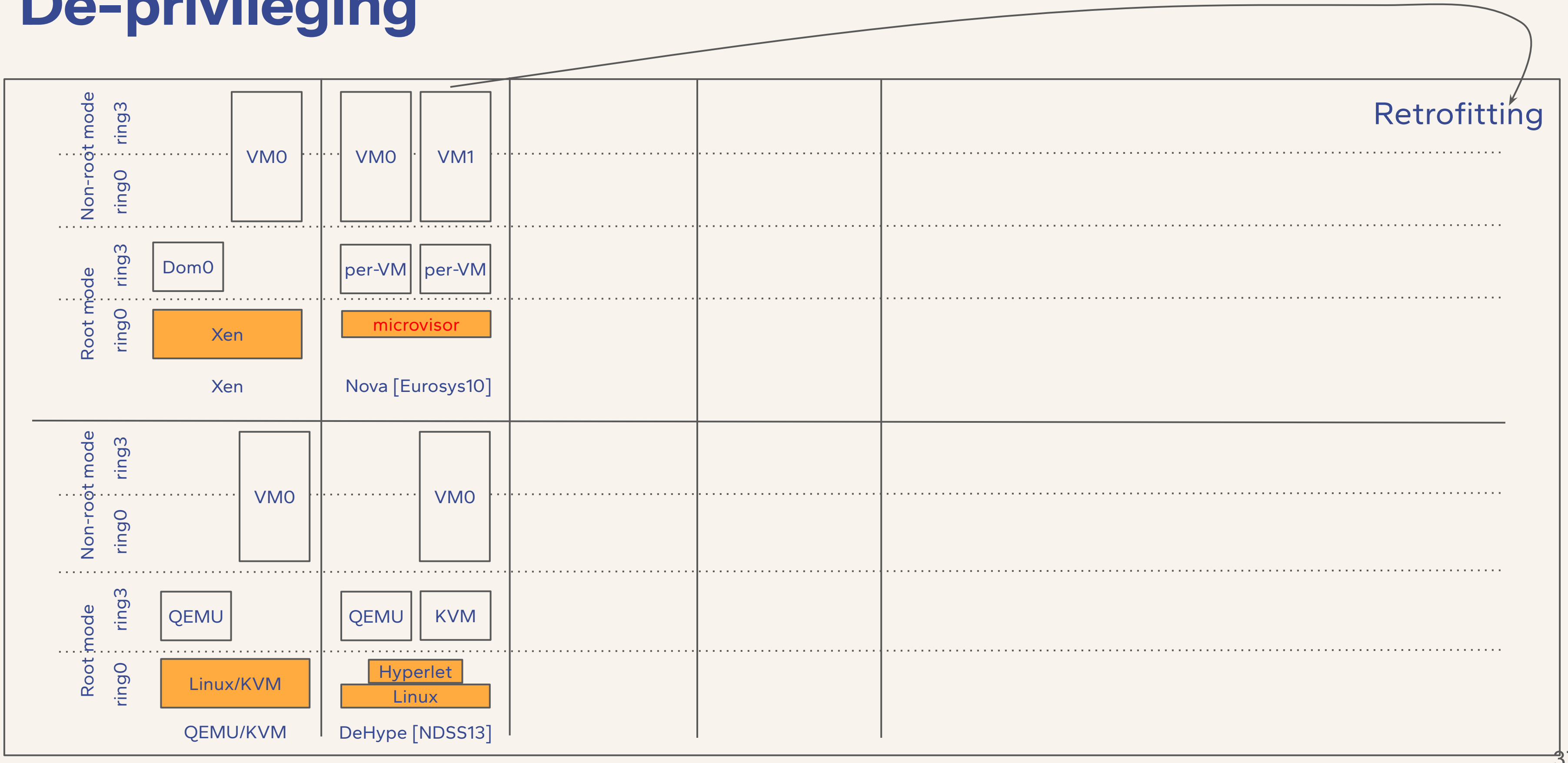
Secure Reimplementation

Apply various techniques to strengthen hypervisor security

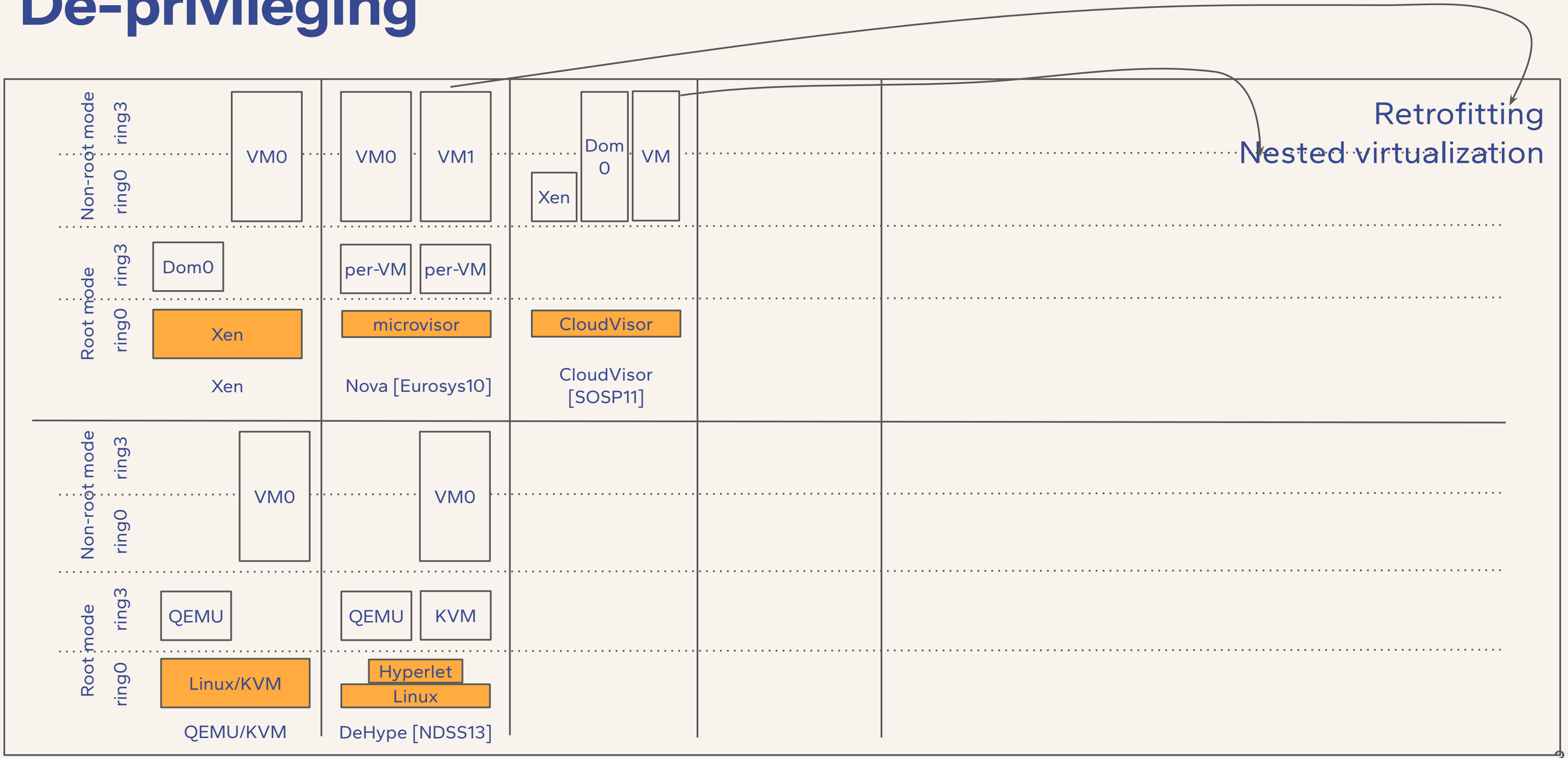
Exploit Prevention

Understand the exploits, detect and prevent them at runtime

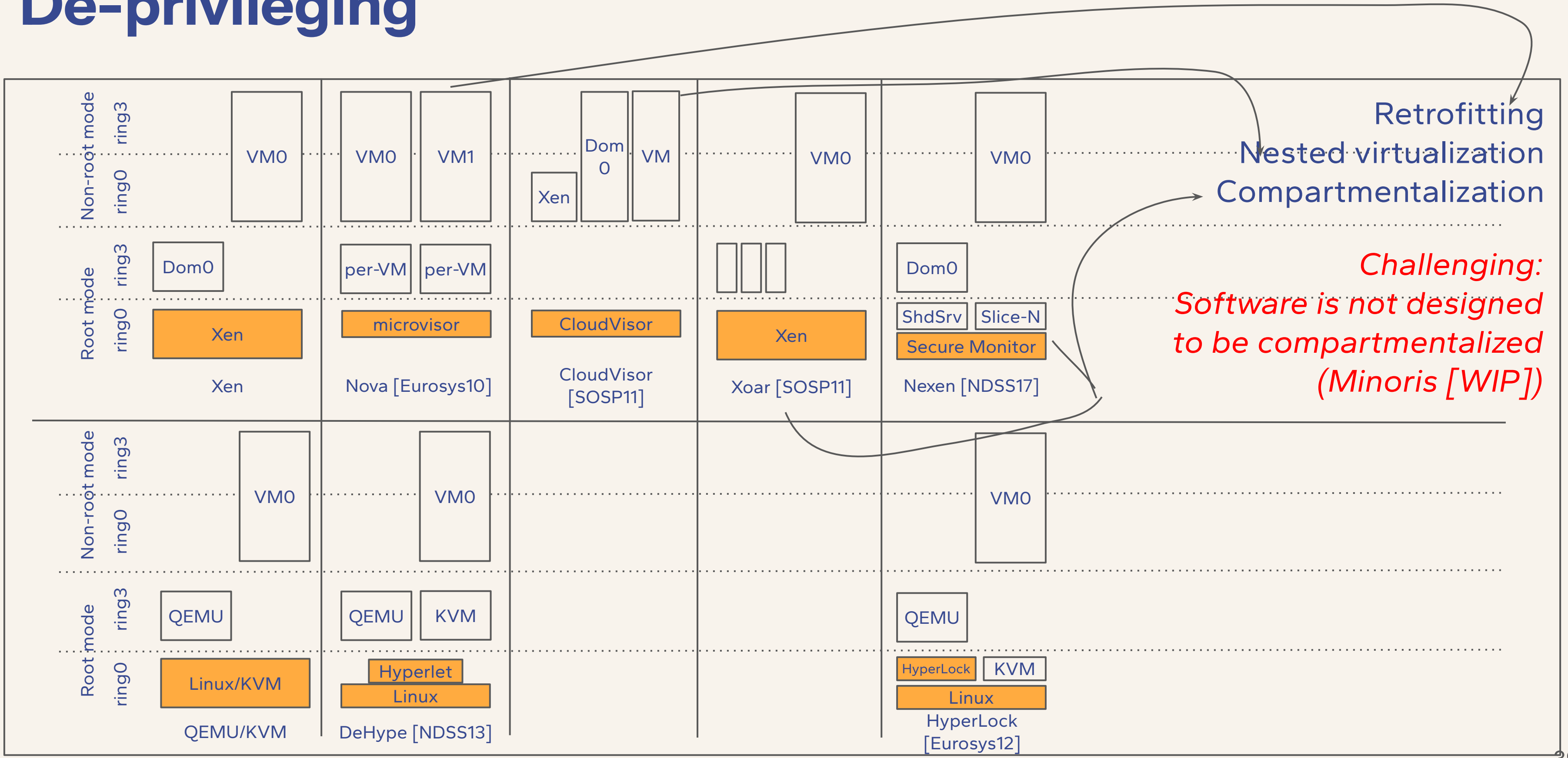
De-privileging



De-privileging



De-privileging



Challenging:
Software is not designed to be compartmentalized
(Minoris [WIP])

Formal Verification SeKVM [S&P21,SOSP21]

Retrofitting enables formal verification

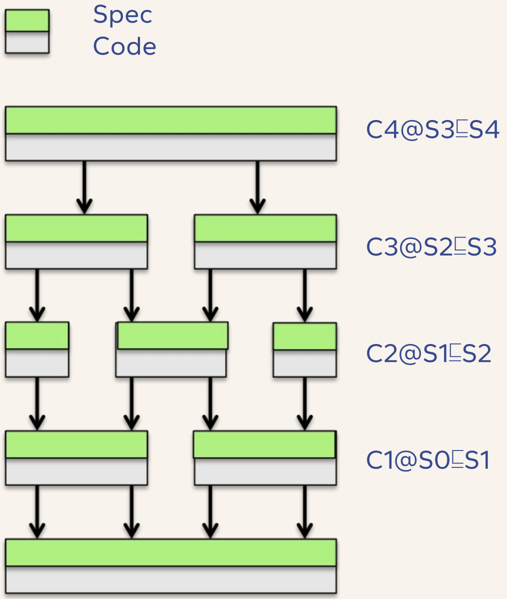
- seL4 (9K LoC): 👷👷👷👷👷👷👷👷👷👷 * 1 year
- CertiKOS (6.5K LoC): 👷👷👷 * 1 year
- SeKVM=retrofit(KVM)=KServ+KCore (3.8K LoC)
    - 👷👷 * 1 year (real workload overhead: <10%)

Step 1: prove the top layer specifies the entire system

Step 2: prove noninterference at the top layer specification

A certain threat model enables the proof of noninterference assertion

- Each VM's data confidentiality and integrity are protected from another VM (concurrency is the key feature to be supported)



Secure Reimplementation

Reimplement hypervisors

Firecracker

- in Rust, e.g., Amazon's Firecracker, KVM-based, *musl libc-based*
    - Started with a branch of Google Chrome's crosvm
    - Very lightweight and fast for multiple-tenant and function-based services
    - A minimum design with 70K LoC of Rust
        - No support of BIOS, Windows, legacy device or PCI, or VM migration
        - Virtual devices: virtio-net/block, serial/keyboard, timers and interrupt controllers
    - Jailer: a wrapper around Firecracker to sandbox it (e.g., chroot, pid/network namespaces, seccomp with 24 whitelist syscalls etc.)

Typical techniques for mitigating attacks include the use of memory-safe programming languages, minimal implementations, sandboxing

Secure Reimplementation

Reimplement hypervisors

- with dedicated hardware, e.g., Amazon's Nitro System
  - Nitro Hypervisor - A KVM-based, firmware-based, and deliberately minimized hypervisor
  - Nitro Cards - Dedicated PCI devices + firmware, with single-root input/output virtualization (SR-IOV) technology, implementing one virtual device with one virtual function
  - Nitro Security Chip — Enabling a secure boot process for the overall system

---

Typical techniques for mitigating attacks include the use of memory-safe programming languages, minimal implementations, sandboxing; decomposition of the software components, secure boot (integrity measurement)

Secure Reimplementation

Reimplement hypervisors

- by exploring architectural features, e.g., Android’s pKVM
  - pKVM enables stage 2 protection in host context
  - pKVM requires IOMMU hardware for every DMA-capable device in the system
  - Use shared bounce buffer for virtio’s data and its metadata
  - Use crosvm that is written in Rust with a few virtual devices, virtio-blk, vhost-vsock, virtio-pci, pl030 real time clock (RTC), and 16550a UART

Typical techniques for mitigating attacks include the use of memory-safe programming languages, minimal implementations, sandboxing; decomposition of the software components, secure boot (integrity measurement); architectural features; finally, it all comes down to trusting KVM!

VM Message Wall to Stop Hypervisor Exploits WIP

Step 1:
Understand

Step 2:
Construct primitives

Step 3:
Exploit

Prevention

VM message sequence

Arbitrary r/w primitives

Information leakage primitives

Control flow primitives

Inject shellcode or reuse existing code

Hijack function pointer (e.g., interrupt handler)

Invoke the hijacked function pointer

VMI: Monitor MMIO/DMA access pattern

Memory Safety: Fine-grained isolation, integrity, and randomization

Outline

Open Questions

Linux/KVM

- How to generate quality input for all VM exits?
- How to detect and prevent race conditions in hypervisors?

Closed-source hypervisors

- How to detect memory corruptions in closed-source hypervisors?
- How to rehost arbitrary cell phone firmware?

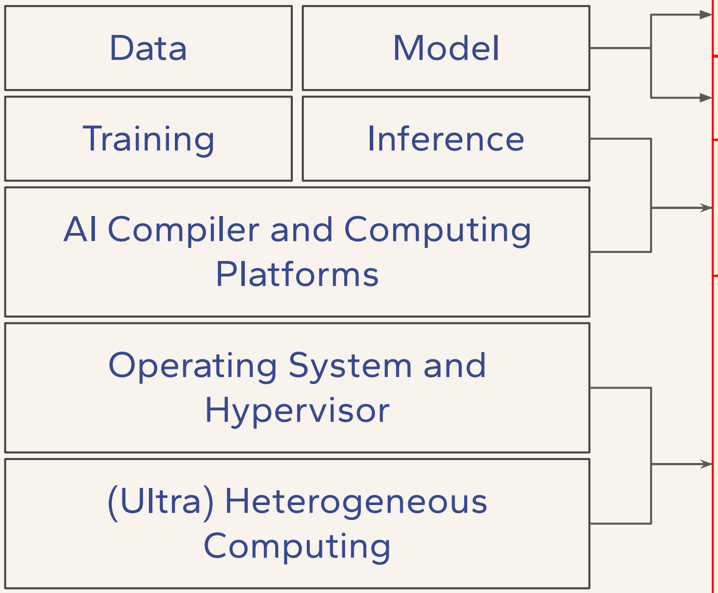
Others

- How to detect logic errors in Rust-based hypervisors?
- How to automatically exploit QEMU/KVM bugs?
- How to test virtio backends?

From System Security to AI System Security

Future Work: AI System Security

It works!!



Open/Online model weights -> Integrity and Confidentiality

Compliance and Collaboration -> Confidential Computing

AI Framework & AI Compilers/Interpreters & AI Hardware Libraries -> Bug findings/Exploitations/Mitigations

Heterogeneous computing -> broken trust boundaries

- Resources scale-up (NVLink, UB-Mesh)
- Resources scale-out (InfiniBand/RoCE, UB-Mesh)
- GPU-centered computing & GPU security
  - Nvidia Triton Inference Server
  - Kernel drivers/Virtualization/Hardware

Future Work: AI System Security

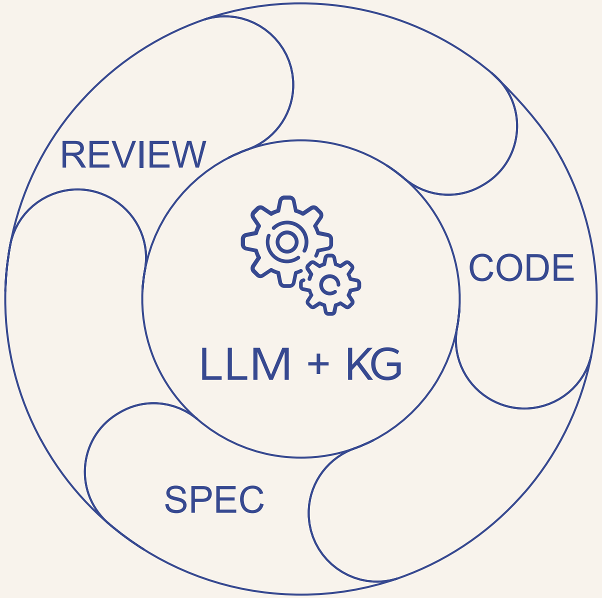
Assets**Lifecycle**	**Pre-training**	**Fine-tuning**	**Inference**
AI Systems	Data -> MW	Data+MW -> MW	Prompts + MW -> Answers
	AI-optimized software stack: DB/AI Compilers/AI Inference Server Specialized hardware for acceleration: GPU/TPU Scalability and distributed computing: xPU/Sharding/Sharing		
	Cloud	Cloud or Local	Cloud or Local/Embedded
Private Data		Shared to LLM via Retrieval-Augmented Generation (RAG)	Shared to LLM via Prompts
Traditional IT Systems			Operated by LLM via Model Context Protocol (MCP)

Future Work: Simplify Low-Level System Understanding

No human can digest

- 14K pages of ARM SPEC
- 10GB reviews of QEMU
- 2M LoC of QEMU
- 29M LoC of Linux kernel
- ...

Code-Survey (LLM for eBPF)
https://arxiv.org/abs/2410.01837



A super model for encoding structured and unstructured knowledge of system software

A super model brings

- Input grammar
- Test coverage insights
- Regression detection
- Crash impact
- Mitigation completeness
- Coding suggestions
- Natural language querying
- Debugging helper
- ...

REVIEW=Code review
SPEC=Specification
LLM=Large Language Model
KG=Knowledge Graph

Future Work: A Formally Verified Limbo

Historical milestones

- Standalone computing (until ~2000)
- Personal computing / Web 2.0 Era (2000–2012)
- Large-scale computing & deep learning (2012-2018)
- <u>Foundation models, AI breakthroughs (2018-current)</u>
- Ubiquitous computing & heterogeneous security era (future)
    - Devices of all forms: personal, enterprise, embedded
    - Edge computing as a global, complex, and distributed fabric
    - Requires unified software ecosystem and security frameworks

Security shift: from defense to resilience

- Success is no longer just about blocking attacks
- Key: fast recovery and business continuity post-incident
- Solution: a thin, scalable, and formally verified minimum recovery system

Conclusion and Q&A

Hypervisor enables Cloud computing by virtualizing and isolating system resources.

Hypervisors are critical and increasingly targeted, as advances in fuzzing have made vulnerabilities easier and cheaper to discover. At the same time, their own security has steadily improved.

Our recent research projects—HyperPill, ViDeZZo, and Truman—enable fuzzing of arbitrary hypervisors with high-quality inputs. However, further investment is needed to enhance their applications.

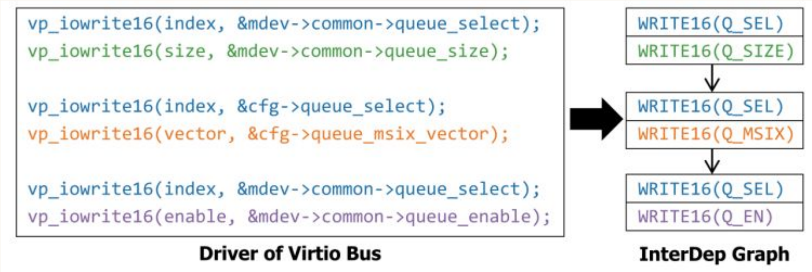
There are various ways to harden hypervisors, but deployment decisions must be cost-effective considering the attacker's return on investment (ROI) and the existing defences.

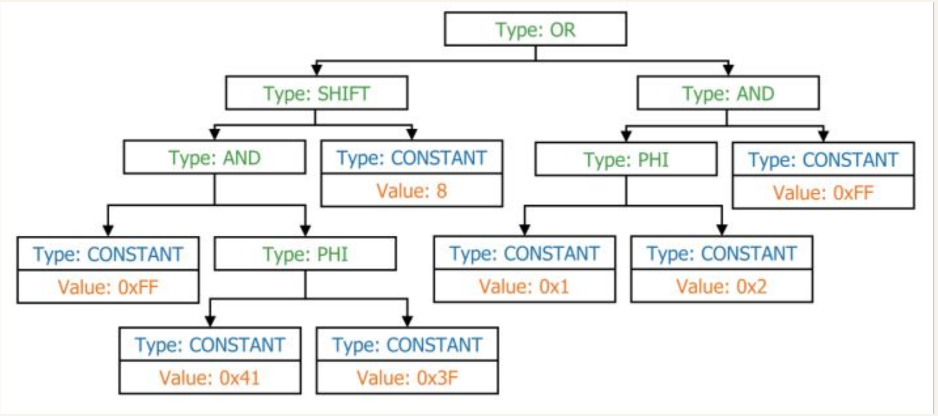
AI introduces not just productivity gains, but also new code, hardware, and usage paradigms—along with fresh vulnerabilities and profit risks. Securing AI systems is more critical than ever!

Contact – Qiang Liu <cyruscyliu@gmail.com>, #opentoconnect

Backup Slides

Static Analysis

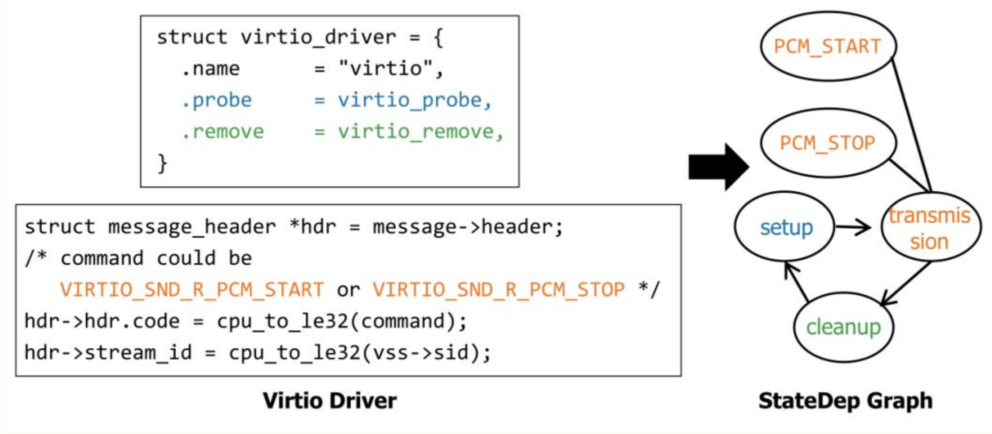




Inter-Message Dependency
CG/CFG Traversal

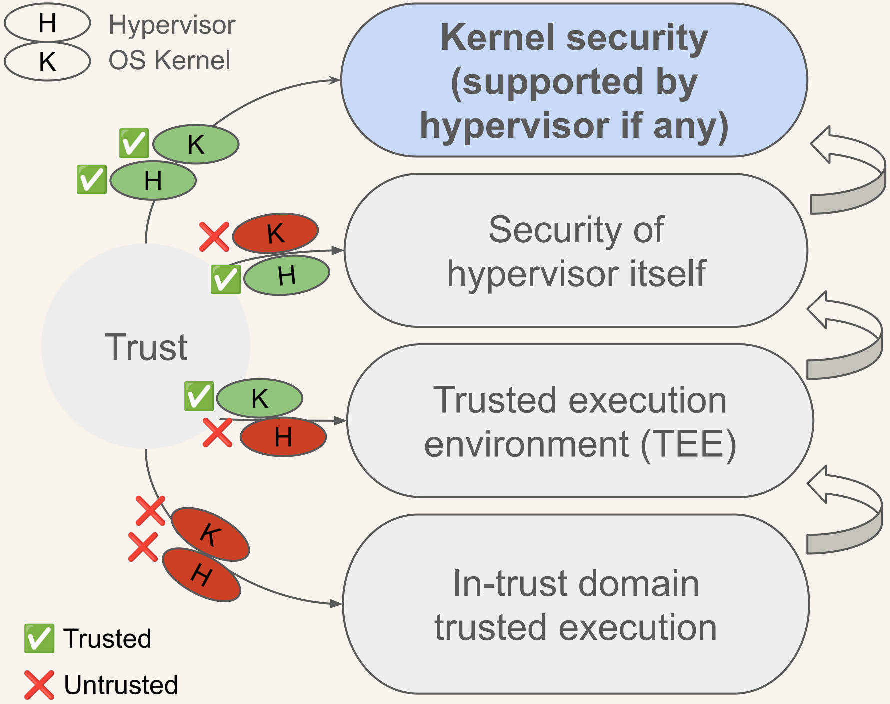
Intra-Message Dependency
Backward Dataflow Analysis

Static Analysis

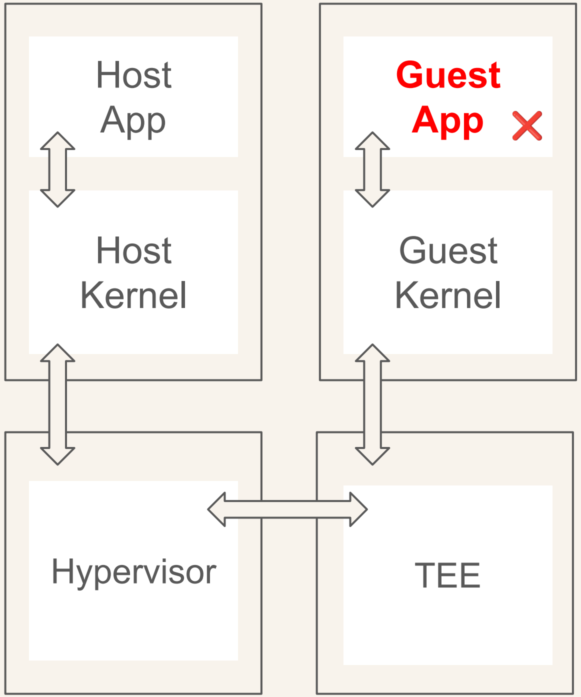


State Dependency
Analyze the bus driver and the device driver

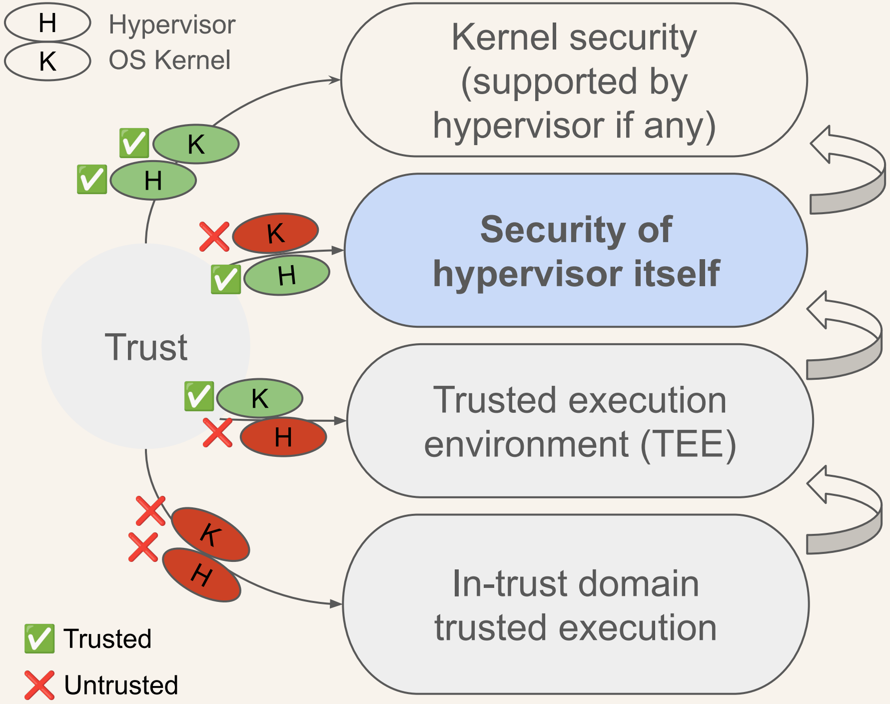
Explicit and transitive trust of the kernel and hypervisor



FirmGuide [ASE21]
ECMO [CCS21]

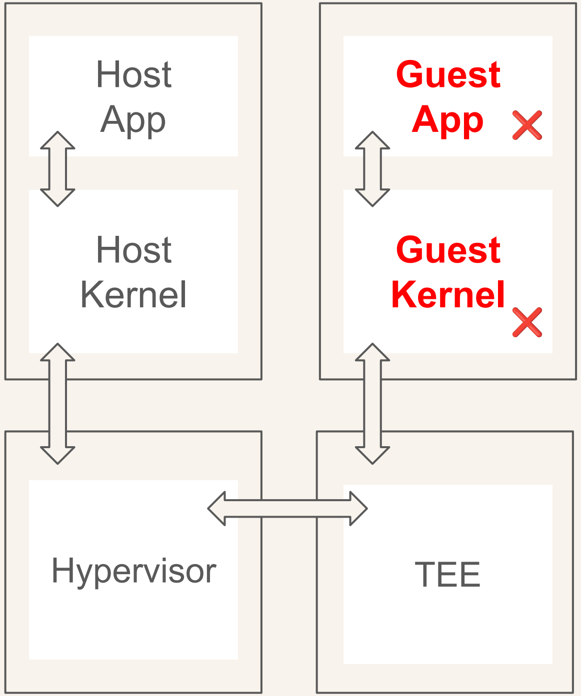


Explicit and transitive trust of the kernel and hypervisor

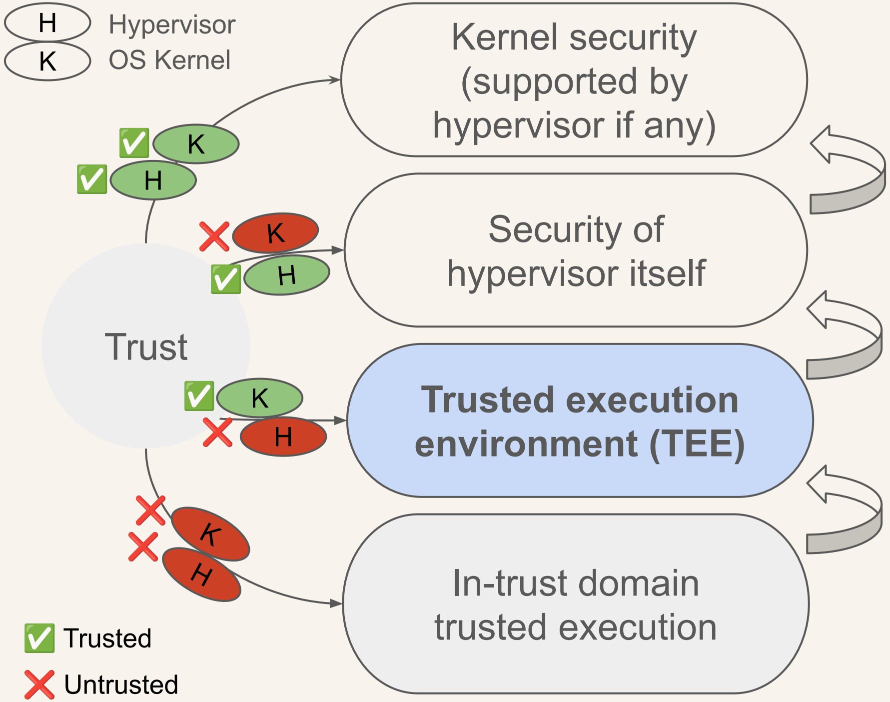


FirmGuide [ASE21]
ECMO [CCS21]

ViDeZZo [SP23]
HyperPill [SEC24]
🏆
Truman [NDSS25]

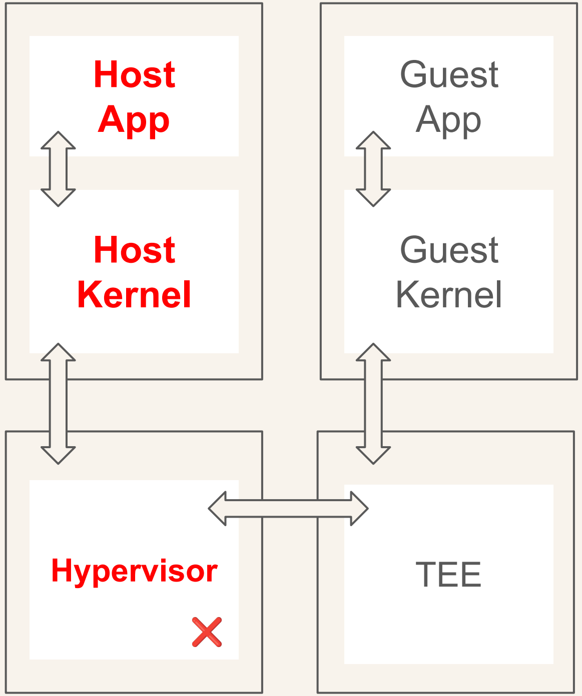


Explicit and transitive trust of the kernel and hypervisor

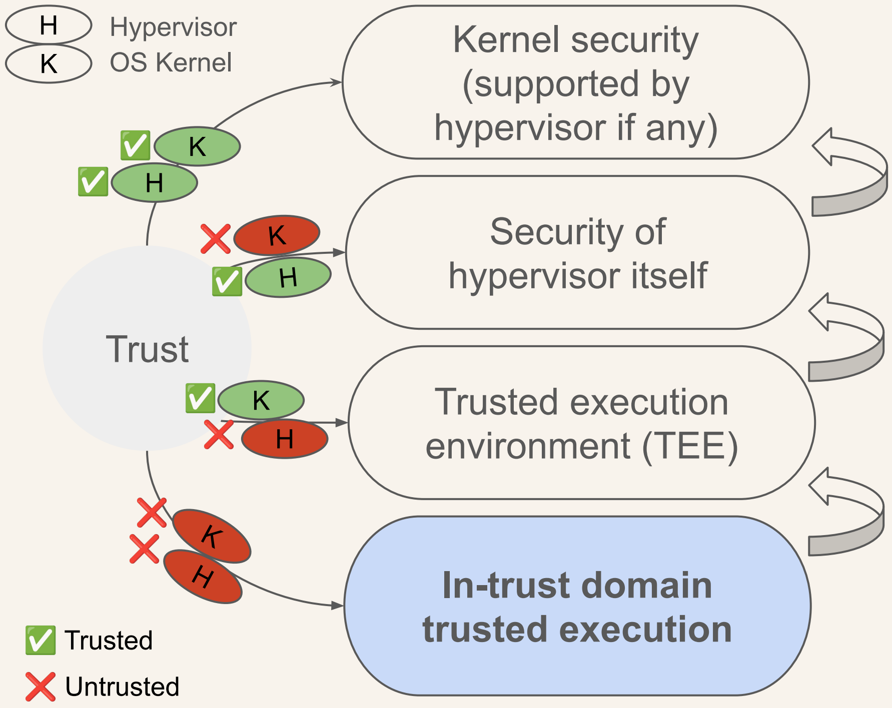


FirmGuide [ASE21]
ECMO [CCS21]

ViDeZZo [SP23]
HyperPill [SEC24]
🏆
Truman [NDSS25]

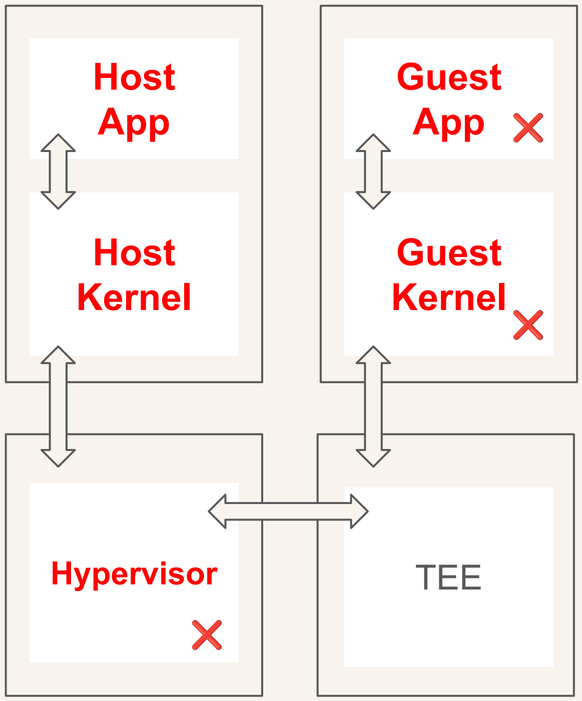


Explicit and transitive trust of the kernel and hypervisor

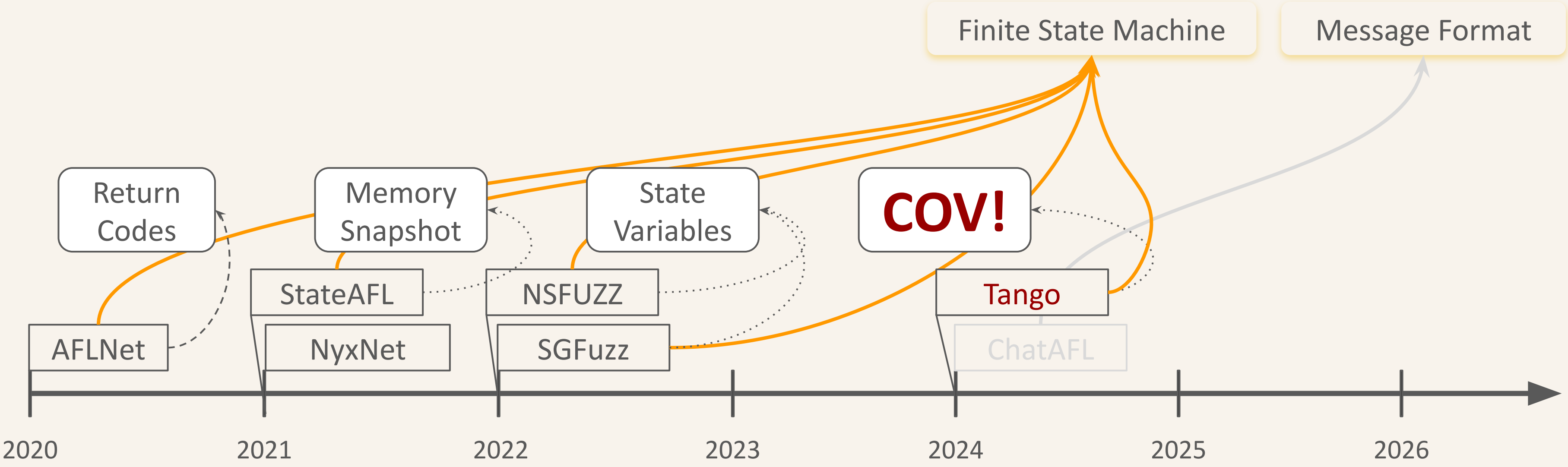


FirmGuide [ASE21]
ECMO [CCS21]

ViDeZZo [SP23]
HyperPill [SEC24]
🏆
Truman [NDSS25]



Tango: Extracting Higher-Order Feedback through State Inference (RAID'24 Best Paper Award)



How can we extract the states in a generic way?

Model-guided kernel execution FirmGuide [ASE21]

How to run a Linux kernel for x86? QEMU!

What about running Linux kernels used in ARM/MIPS-based IoT devices?

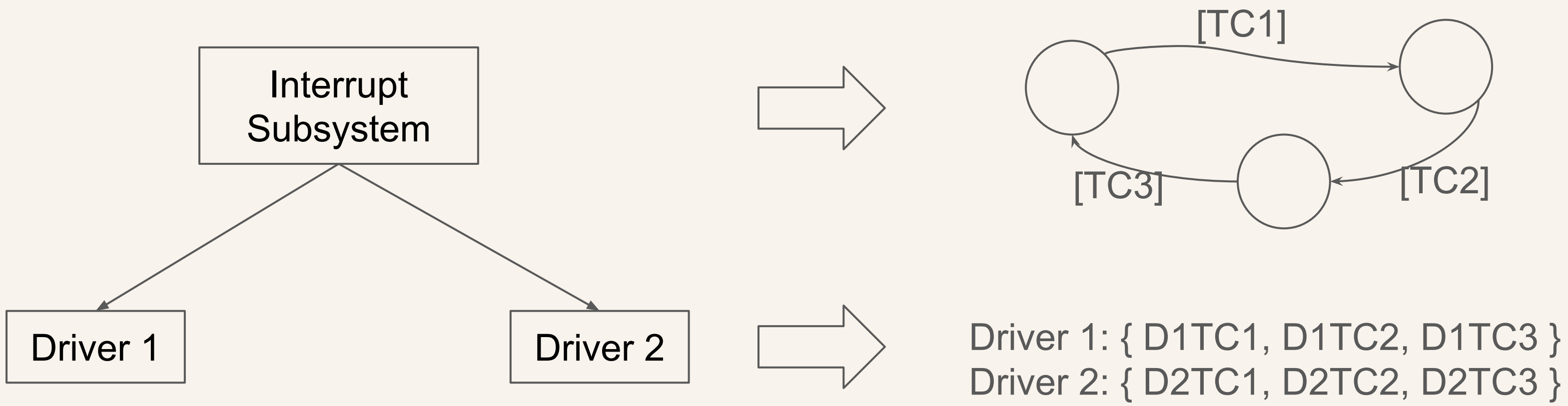
- Challenges: ARM/MIPS devices have fragmented peripherals
- Aim at a minimum best effort to boot with an interactive shell [FirmGuide ASE21]

ARM chip example: plxtech,nas782x		QEMU	Fidelity for booting
CPU	Arm11MPCore	✅	High
Memory	up to 512M	✅	High
Interrupt controller	plxtech,nas782x-rps	❌	High
Time-related	rps-timer, oscillator, sysclk, plla, pllb, stdclk, twdclk	❌	High
UART	ns16550a	✅	High
Other peripherals	gmacclk, pcie, watchdog, sata, nand, ethernet, ehci, leds	❌	Low

Model-guided kernel execution FirmGuide [ASE21]

Linux kernel subsystem defines a state machine driven by driver behavior

A peripheral model = a state machine + driver behavior as transition conditions



Model-guided kernel execution FirmGuide [ASE21]

It starts

The Linux Kernel

QEMU libvirt KVM

State 1

Model-guided kernel execution FirmGuide [ASE21]

Our peripheral model is at state 1 and have monitored the behavior of the Linux kernel, specifically by logging MMIO rw sequences (MMIO R/W Seqs)

Model-guided kernel execution FirmGuide [ASE21]

Our peripheral model goes to state 2 if the MMIO R/W Seq matches D1TC1

Model-guided kernel execution FirmGuide [ASE21]

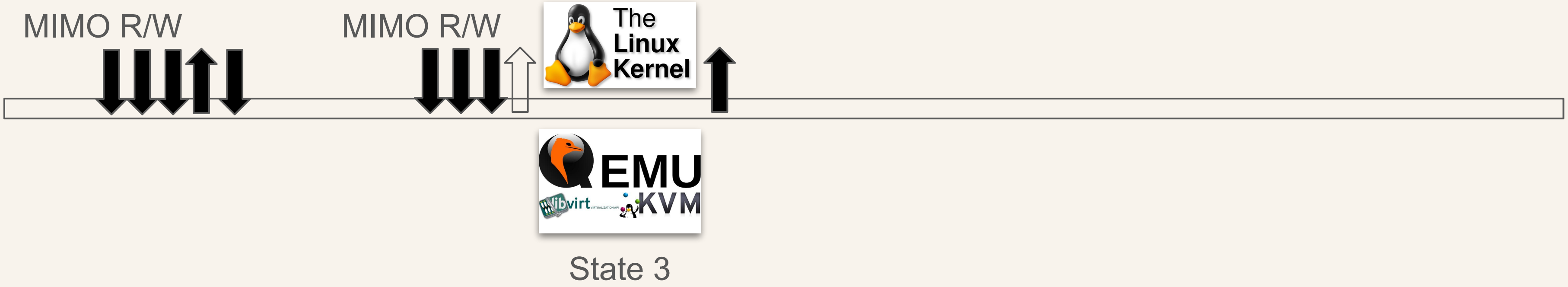
Linux kernel runs

Model-guided kernel execution FirmGuide [ASE21]

Our peripheral model is at state 2 and have monitored another MMIO R/W Seq

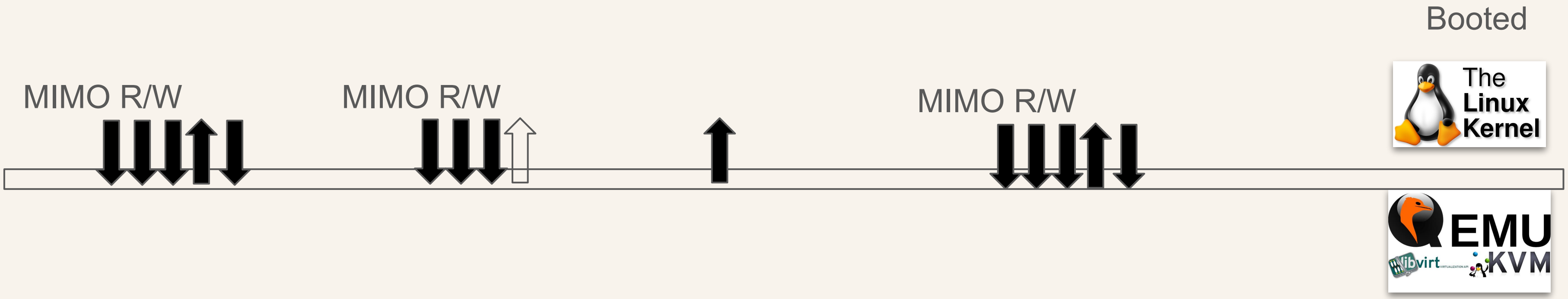
Model-guided kernel execution FirmGuide [ASE21]

Our peripheral model goes to state 3 with a value back



Model-guided kernel execution FirmGuide [ASE21]

Until we get an interactive shell



Model-guided kernel execution FirmGuide [ASE21]

Techniques

- Use KLEE to extract MMIO R/W Seqs from Linux kernel drivers
- Use a template render to composite a QEMU machine

Results

- We first enabled the fuzzing of embedded Linux kernels for 26 SoCs
- We managed to develop exploits, which can never be easily done without successful rehosting.
- We showed that backporting kernel patches for IoT devices was not yet timely.