Towards Full-Lifecycle Security Enforcement of Hypervisors

Qiang Liu, PostDoc@EPFL

EPFL

hexhive

Outline

Introduction to Hypervisors

A Full-Lifecycle Enforcement of System Security

Hypervisors: Ahead-of-Release Bug Fixes

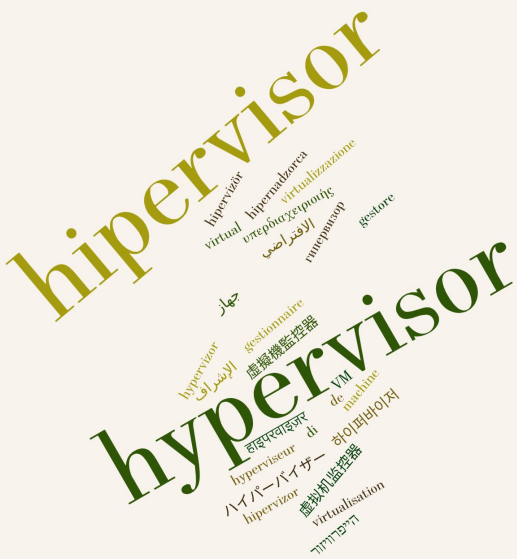
Hypervisors: In Production Attack Mitigation

Open Questions and Future Work

A Predestined Journey to the Cloud

A friend of mine is scaling his AI workloads ...

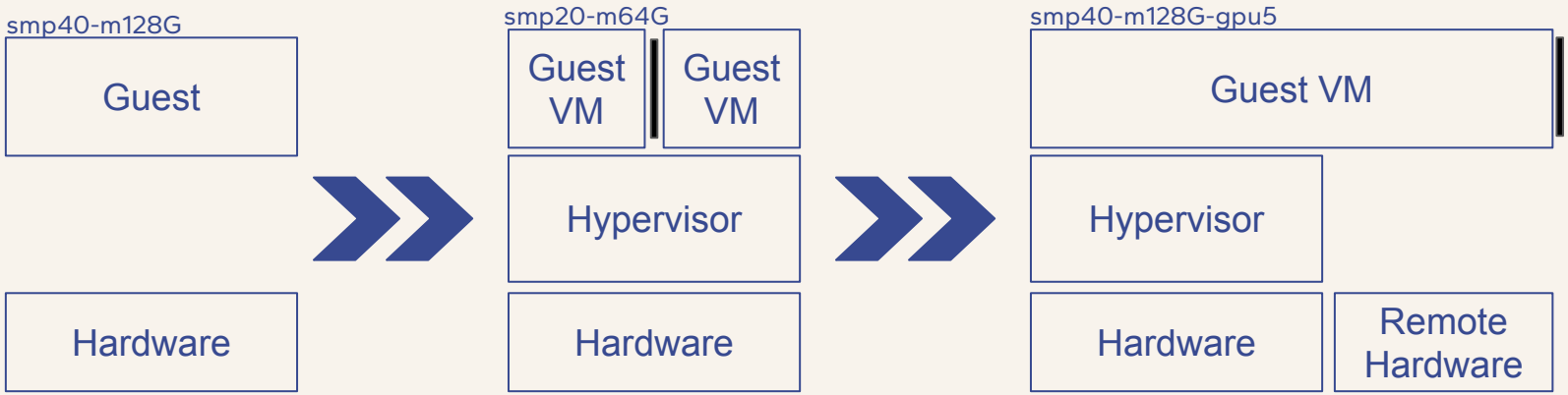
- 💵 **Develop** an AI-powered service
- 😎 **Have** rapidly increasing demand
- 😫 **Struggle** with buying, shipping, and setup physical hardware. TIME IS MONEY!
- 🤔 **Require** AI workloads to be on-demand, scalable, and cost-efficient
- 💵💵💵 **Leverage the Cloud** for on-demand resources, but how is this possible?




Hypervisor.Framework - Virtualization on macOS

Hypervisors: Logical Concepts and Core Isolation

Hypervisors create virtual machines



No hypervisor
the guest controls hardware directly

With hypervisor
the guest appears to control hardware

With pooling
the guest can scale resources (e.g., GPUs) as needed for workloads like AI

Hypervisors: Attacker's Return on Investment (ROI)

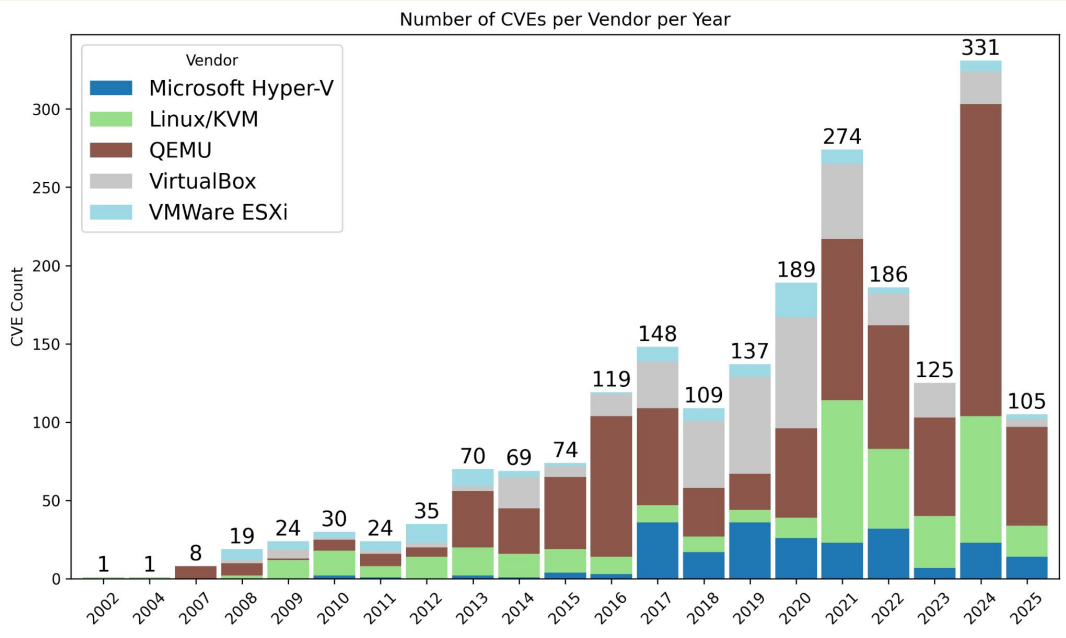
Cost↘

QEMU/KVM CVEs on the rise

- 🏆 Due to fuzzing

Hyper-V/ESXi CVEs consistently

- ✅ Closed-source
- 🙊 Internal fixes
- 😭 Lack of sanitizers



Hypervisors: Attacker's Return on Investment (ROI)

Gain

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

VULNERABILITIES

VENOM Vulnerability Opens Millions of Virtual Machines to Attack

May 18, 2015 by Pierluigi Paganini

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.

Outline

Introduction to Hypervisors

A Full-Lifecycle Enforcement of System Security

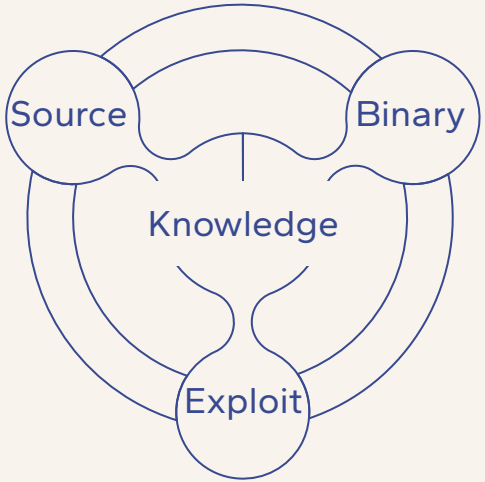
Hypervisors: Ahead-of-Release Bug Fixes

Hypervisors: In Production Attack Mitigation

Open Questions and Future Work

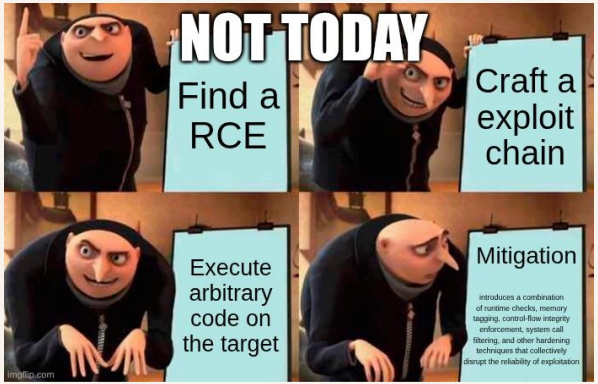
A Full-Lifecycle Enforcement of System Security

Ahead-of-release bug fixes



Exploitation as evaluation

In-production attack mitigation



A Full-Lifecycle Enforcement of System Security

A PhD isn't about solving everything—
it's about solving one hard problem really well.

In this talk, I'll ***focus on the ahead-of-release bug finding part***, while leaving many exciting directions—such as in production mitigation, exploit-based evaluation, and ***research beyond hypervisors*—for future work**.

Outline

Introduction to Hypervisors

A Full-Lifecycle Enforcement of System Security

Hypervisors: Ahead-of-Release Bug Fixes

Hypervisors: In Production Attack Mitigation

Open Questions and Future Work

Hypervisors: Ahead-of-Release Bug Fixes

Fuzzing: the most widely adopted and effective approach for bug discovery

Threat model: the guest VM is not trusted; the attacker has the root privilege

Two research questions:

Execution Environment

How to drive arbitrary hypervisors in a ***unified framework***?

Input Generation

How to generate ***high-quality*** inputs for hypervisor testing?

Hypervisors: Ahead-of-Release Bug Fixes

Fuzzing: the most widely adopted and effective approach for bug discovery

Threat model: the guest VM is not trusted; the attacker has the root privilege

Two research questions:

Execution Environment

How to drive arbitrary hypervisors in a ***unified framework***?

Hyper-Cube[NDSS20]
Nyx[SEC21]
Morphuzz [SEC22]
*HyperPill[SEC24]🏆
*HyperARM

Input Generation

How to generate ***high-quality*** inputs for hypervisor testing?

V-Shuttle [CCS21]
Morphuzz [SEC22]
MundoFuzz [SEC22]
*ViDeZZo [S&P23]
*Truman [NDSS25]

An asterisk (*) denotes my own work.

A snapshot-based Hypervisor Dock HyperPill [SEC24]

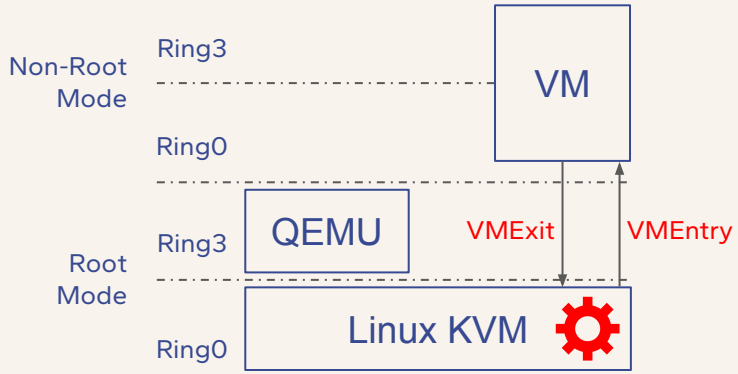
What do all the following hypervisors have in common?

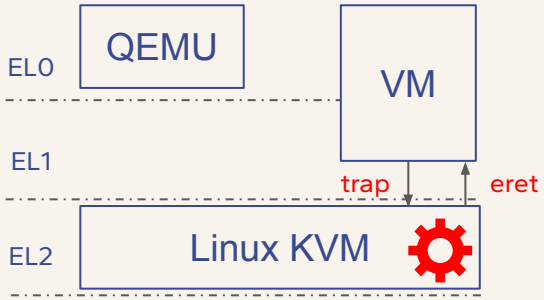
- All hypervisors implement the same hardware virtualization interface

A snapshot-based Hypervisor Dock HyperPill [SEC24]

What do all the following hypervisors have in common?

- All hypervisors implement the same hardware virtualization interface
- Trap-and-emulate: execute most guest instructions natively on hardware but trap and emulate "some" instructions, e.g., in/out (x86), mrs/msr (arm)

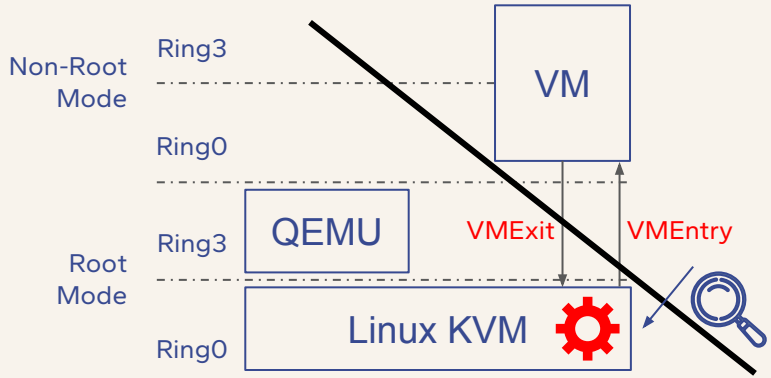


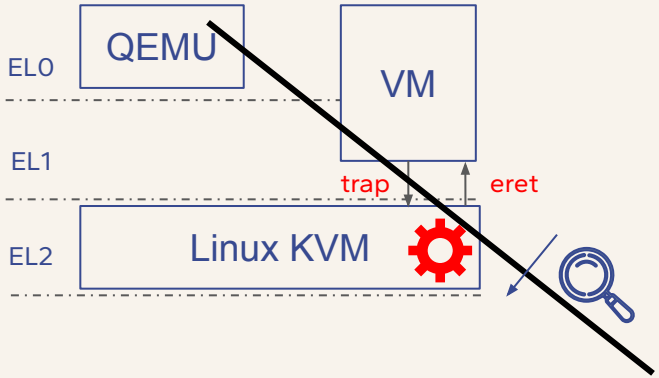


A snapshot-based Hypervisor Dock HyperPill [SEC24]

What do all the following hypervisors have in common?

- All hypervisors implement the same hardware virtualization interface
- Trap-and-emulate: execute most guest instructions natively on hardware but trap and emulate "some" instructions, e.g., in/out (x86), mrs/msr (arm)

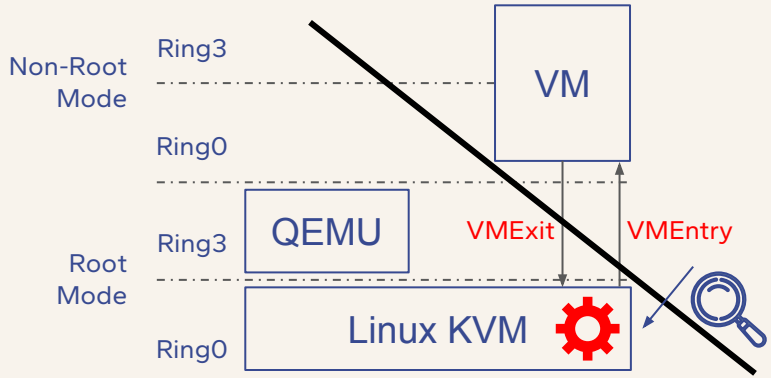


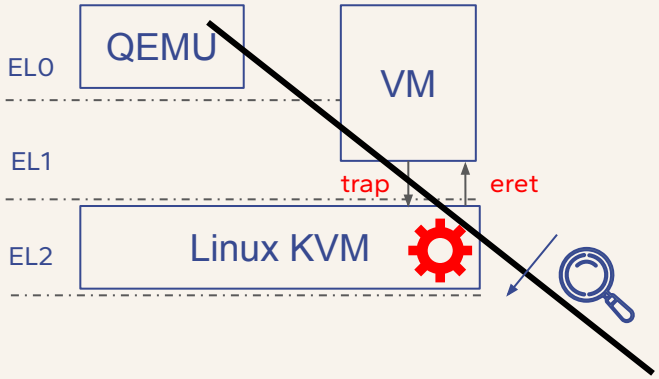


A snapshot-based Hypervisor Dock HyperPill [SEC24]

What do all the following hypervisors have in common?

- All hypervisors implement the same hardware virtualization interface
- Trap-and-emulate: allow us to have a unified view of the hypervisor

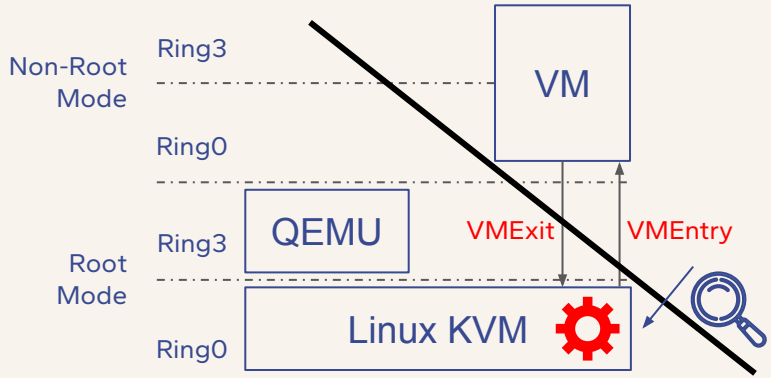


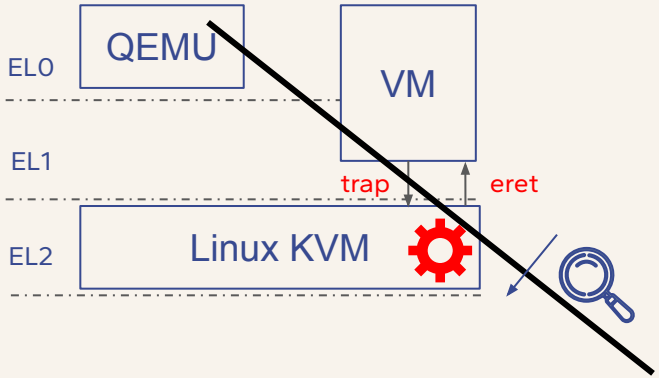


A snapshot-based Hypervisor Dock HyperPill [SEC24]

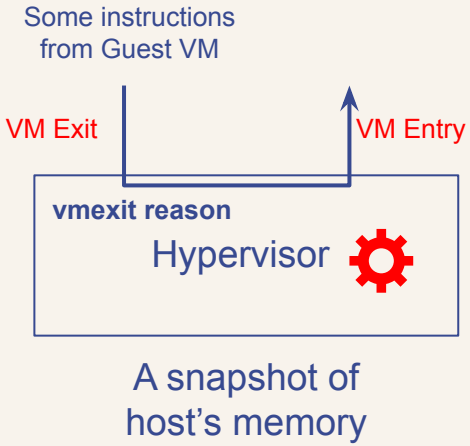
What do all the following hypervisors have in common?

- All hypervisors implement the same hardware virtualization interface
- Trap-and-emulate: allow us to have a unified view of the hypervisor
- Let's snapshot the host memory and inject inputs (via "some" instructions) into it





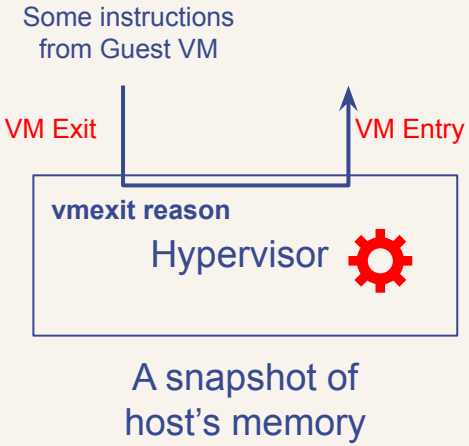
A snapshot-based Hypervisor Dock HyperPill [SEC24]



A snapshot-based Hypervisor Dock HyperPill [SEC24]

Input/VM message that a hypervisor can take

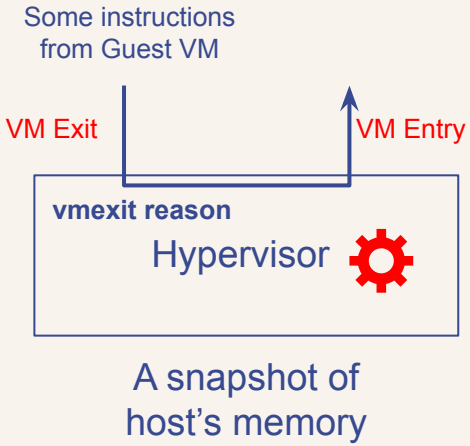
- **Port I/O (PIO):** in/out instructions (x86 only)
- **Memory-Mapped I/O (MMIO)**: memory load/store instructions, e.g., mov (x86), ld/st (ARM)
- **Access to control and status registers**: rdmsr/wrmsr (x86) and mrs/msr (ARM)
- **Hypercalls**: vmcall (x86) or hvc (ARM)
-
- **Access to prefilled memory for DMA requests**: Virtual devices may read from or write to guest memory regions pre-filled with input data via DMA



A snapshot-based Hypervisor Dock HyperPill [SEC24]

Four steps to drive a hypervisor to run

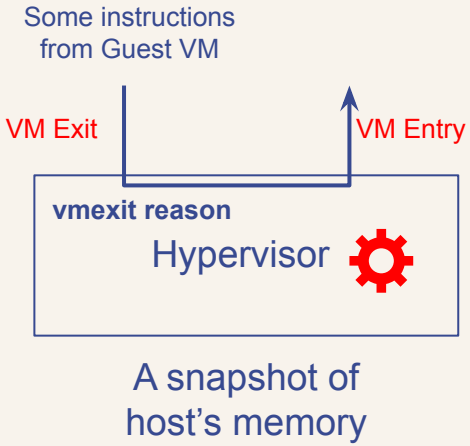
- Modify the vmexit reason and its parameters
  - For x86, the vmexit reason and its parameters (named qualification) are stored in an memory object named Virtual Machine Control Structure (VMCS)
  - For ARM64, the vmexit reason and its parameters are stored in different system registers such as ESR_EL2, FAR_EL2, HPFAR_EL2



A snapshot-based Hypervisor Dock HyperPill [SEC24]

Four steps to drive a hypervisor to run

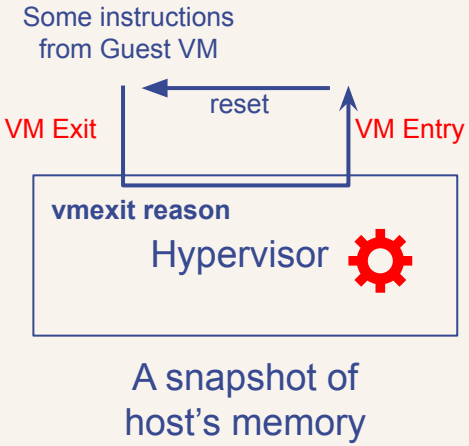
- Modify the vmexit reason and its parameters
  - VMCS (x86), ESR/FAR/HPFAR_EL2 (ARM)
- Let the hypervisor run to process this vm message
  - Wait and stop at the VM-Entry
  - For x86, it's vmresume
  - For ARM64, it's the eret to EL1



A snapshot-based Hypervisor Dock HyperPill [SEC24]

Four steps to drive a hypervisor to run

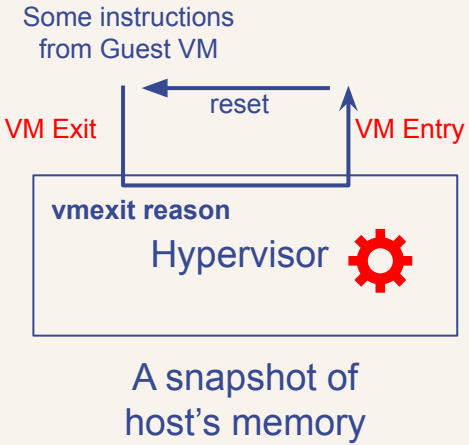
- Modify the vmexit reason and its parameters
  - VMCS (x86), ESR/FAR/HPFAR_EL2 (ARM)
- Let the hypervisor run to process this *vm message*
  - vmresume (x86), eret to EL1 (ARM)
- Reset PC to a vmexit after each vm message
- Reset the whole snapshot if done
  - All system registers, dirty pages



Knowledge-based Input Generation ViDeZZo [SP23] Truman [NDSS25]

A typical sequence of vm messages

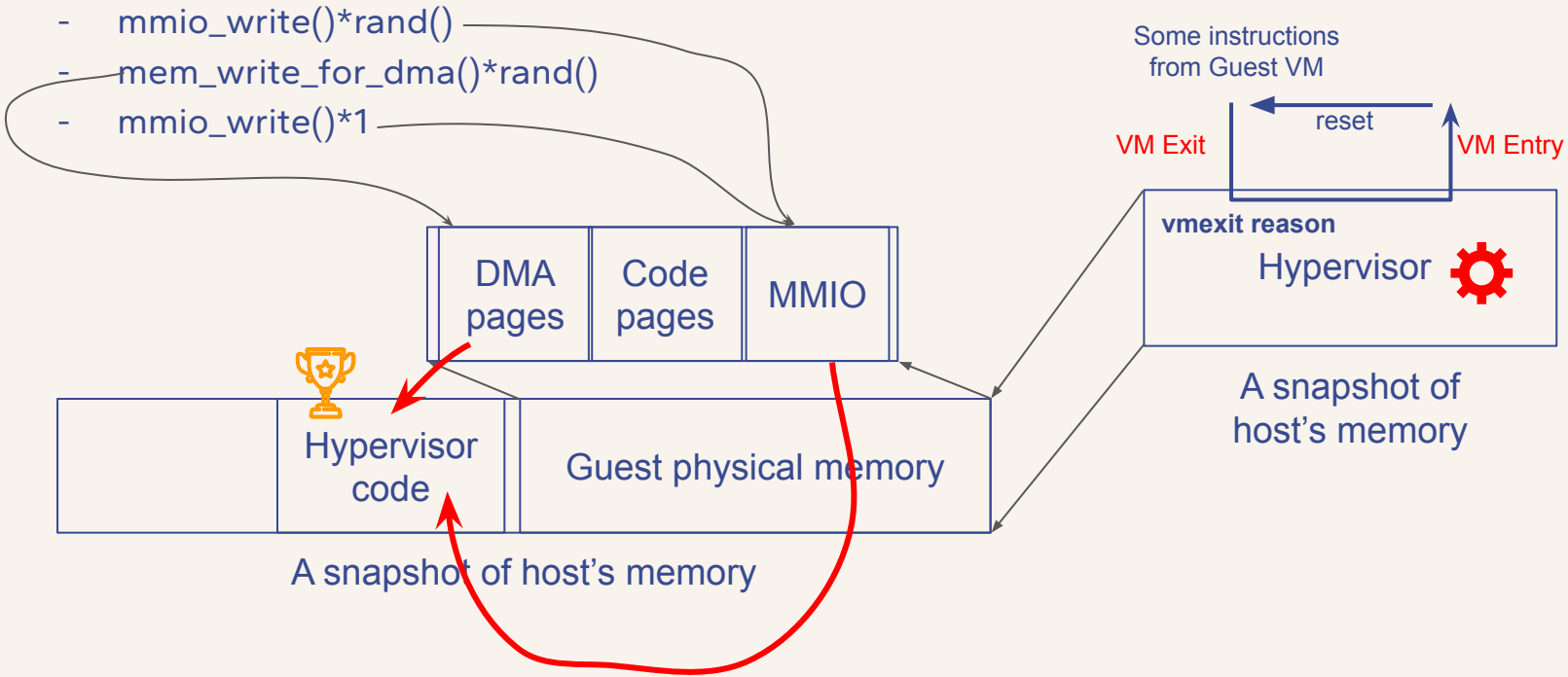
- mmio_write()*rand()
- mem_write_for_dma()*rand()
- mmio_write()*1



A snapshot of
host's memory

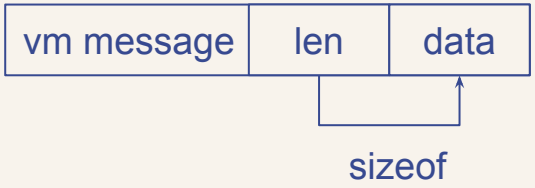
Knowledge-based Input Generation ViDeZZo [SP23] Truman [NDSS25]

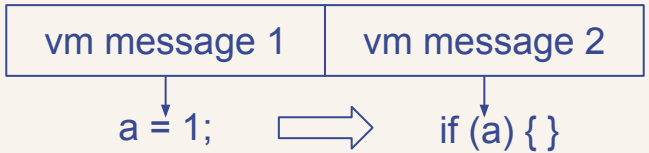
A typical sequence of vm messages

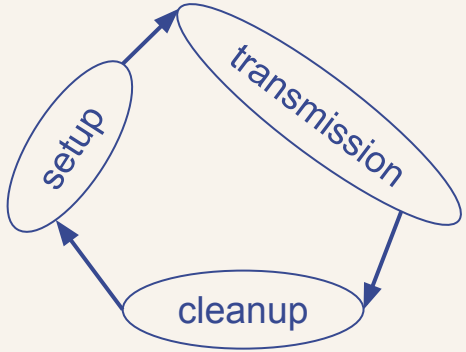


Knowledge-based Input Generation ViDeZZo [SP23] Truman [NDSS25]

Three dependencies







Intra-message dependency

A field in a message may be dependent on another field

Inter-message dependency

A field in a message may be dependent on another field

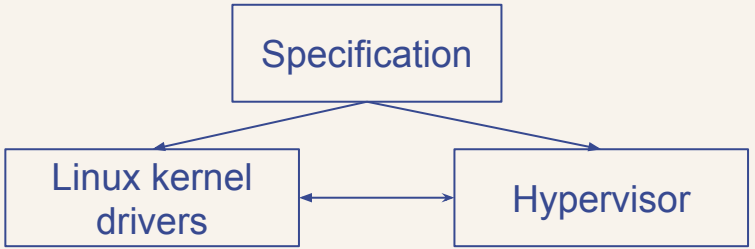
State dependency

A (bus-hidden) component follows a finite state machine

Knowledge-based Input Generation ViDeZZo [SP23] Truman [NDSS25]

Automatic extraction of three dependencies

- Knowledge is encoded in different formats
- From hypervisor code, hard
  - Open-source requirement, no abstraction
- From the Linux Kernel drivers, easier
  - Mostly open-source, with abstraction



Results of Hypervisor Fuzzing

- With ViDeZZo/Truman/HyperPill, we found 108 (54 due to Truman) security bugs covering 6 hypervisors, QEMU/VBox/Hyper-V/macOS virtualization framework/VMware/Parallels
- We manage to extract three kinds of dependencies for 29 virtual devices (including virtio), covering various categories, i.e., audio, storage, network, display, and USB
- HyperPill is the first tool to analyze arbitrary x86 (now extended to ARM64) hypervisors across all major attack-surfaces (i.e., PIO/MMIO/Hypercalls/DMA)

Insights from bug findings for mitigations

- MMIO can be re-entered via DMA
- Hijacked IRQ handlers can be easily triggered
- Sub-page metadata needs fine-grained protection

Outline

Introduction to Hypervisors

A Full-Lifecycle Enforcement of System Security

Hypervisors: Ahead-of-Release Bug Fixes

Hypervisors: In Production Attack Mitigation

Open Questions and Future Work

Hypervisors: In Production Attack Mitigation

Retrofitting and De-privileging

Adapt existing hypervisor code to enforce the principle of least privilege

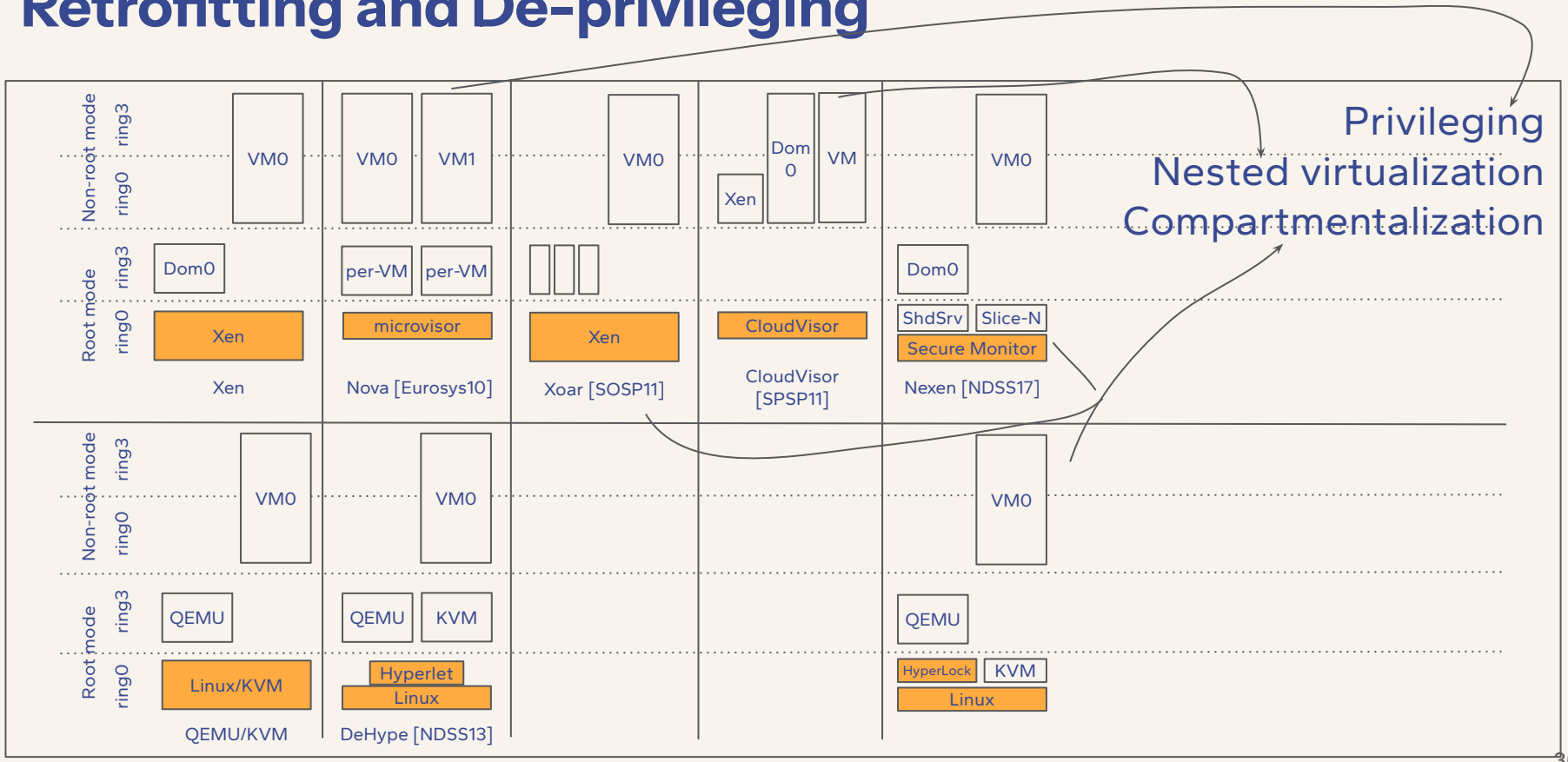
Formal Verification

Adapt an existing hypervisor for verification against security properties

Secure Reimplementation

Apply various techniques to strengthen hypervisor security

Retrofitting and De-privileging



Formal Verification SeKVM [S&P21,SOSP21]

Retrofitting enables formal verification

- It took seL4 ten person-years to verify 9K LoC
- It took CertiKOS three person-years to verify 6.5K LoC
- It took SeKVM one person-year to retrofit KVM (2M LoC) to KServ and KCore (3.8K LoC) and two person-years to complete the verification
    - KVM unit tests: 17%-28% overhead, Real application workload: less than 10% overhead

---

Formal verification has not scaled to large codespace

Formal Verification SeKVM [S&P21,SOSP21]

A careful definition of the threat model enables one kind of proof

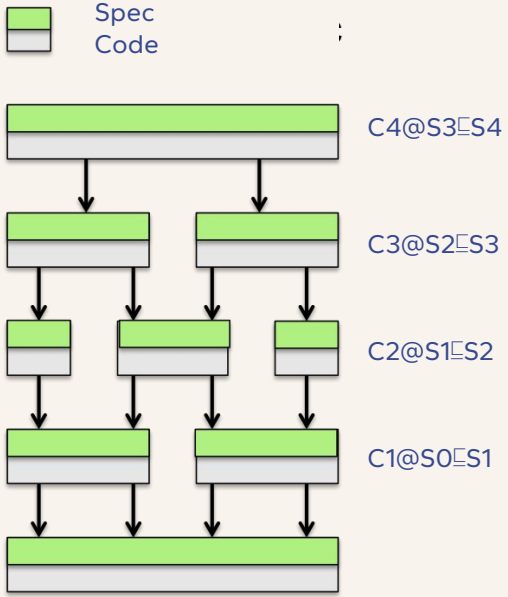
- Each VM's data confidentiality and integrity are protected from another VM
  - i.e., other data and code are not protected, availability is not guaranteed
  - and concurrency is the key feature to be verified
- KServ/Other VMs may be controller by attackers
  - i.e., guest root privilege, access of guest physical memory, abuse of KCore's interfaces exposed to KServ/VM, running on the same machine, running one or more CPUs
- Side-channel attacks are not considered

Proof of data confidentiality and integrity can be converted into the proof of noninterference assertion, which is well defined

Formal Verification SeKVM [S&P21,SOSP21]

The game is on!

- Convert C/ASM implementation to Coq representation
- Write Coq specs by defining security-preserving layers
  - Each implementation refines its interface specification
  - The layer refinement relation is transitive and therefore the top layer specifies the entire system
- Prove the top layer satisfied with noninterference assertions considering concurrency, i.e., prove that KServ and VM won't break other's VM's data confidentiality and integrity



Secure Reimplementation

Reimplement hypervisors

- **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!

Outline

Introduction to Hypervisors

A Full-Lifecycle Enforcement of System Security

Hypervisors: Ahead-of-Release Bug Fixes

Hypervisors: In Production Attack Mitigation

Open Questions and Future Work

Open Questions

- How to generate quality input for all VM exits?
- How to detect memory corruptions in closed-source hypervisors?
- How to detect logic errors in Rust-based hypervisors?
- How to rehost arbitrary cell phone firmware?
- How to detect and prevent race conditions in hypervisors?
- How to automatically exploit QEMU/KVM bugs?
- How to prevent QEMU/KVM exploits in the wild?

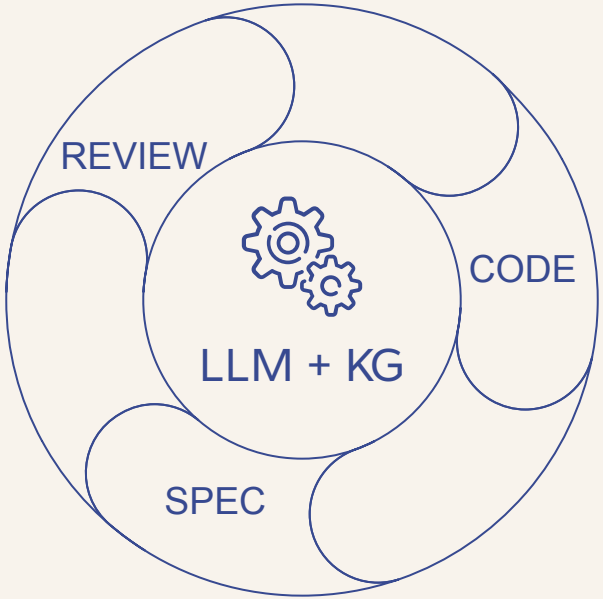
Hypervisor research also yields broadly applicable techniques

Future Work 1: Knowledge is Power!

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 in 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 2: AI Infrastructure Security

A friend of mine is scaling his AI workloads ...

- Uploading model weights that must not be exposed to the cloud provider
  - How to prevent online model weights from being stolen?
- Demanding more than one machine can offer, at the cost of system isolation
  - How to adopt OS/hypervisors to remote resources, e.g., GPU, memory?
- GPU! GPU! GPU! But no proper memory protection, and attack-prone software all around
  - How to stop the memory war of GPU?

Conclusion and Q&A

Hypervisors create virtual machines and are under attacks

Full-lifecycle enforcement of system security requires

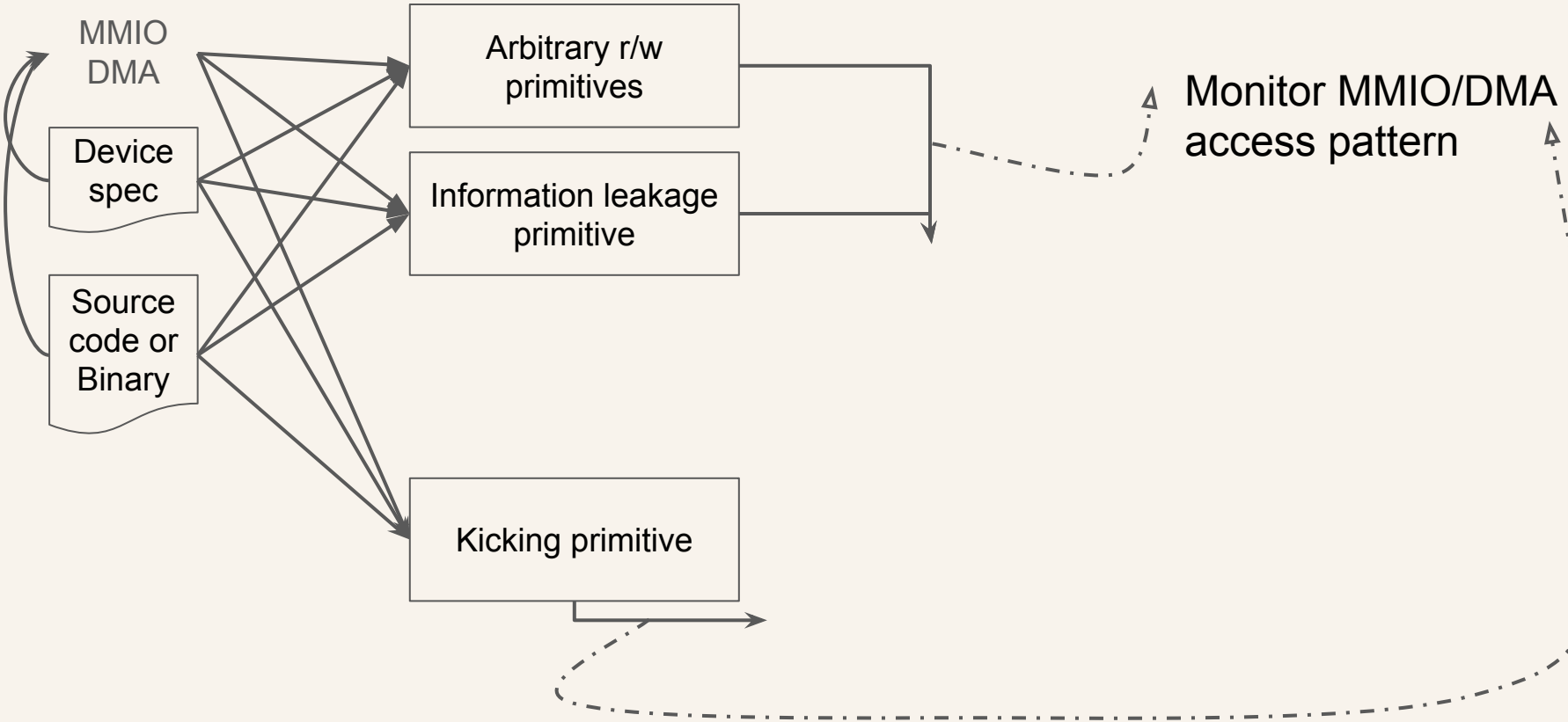
- Ahead-of-release bug fixes and in production attack mitigation
- We contribute to hypervisor fuzzing with a snapshot-based hypervisor dock and definition and extraction vm message dependencies
- Retrofitting and de-privileging, formal verification, and secure reimplementation are used to mitigate attacks to hypervisors

Find me at EPFL: BC154

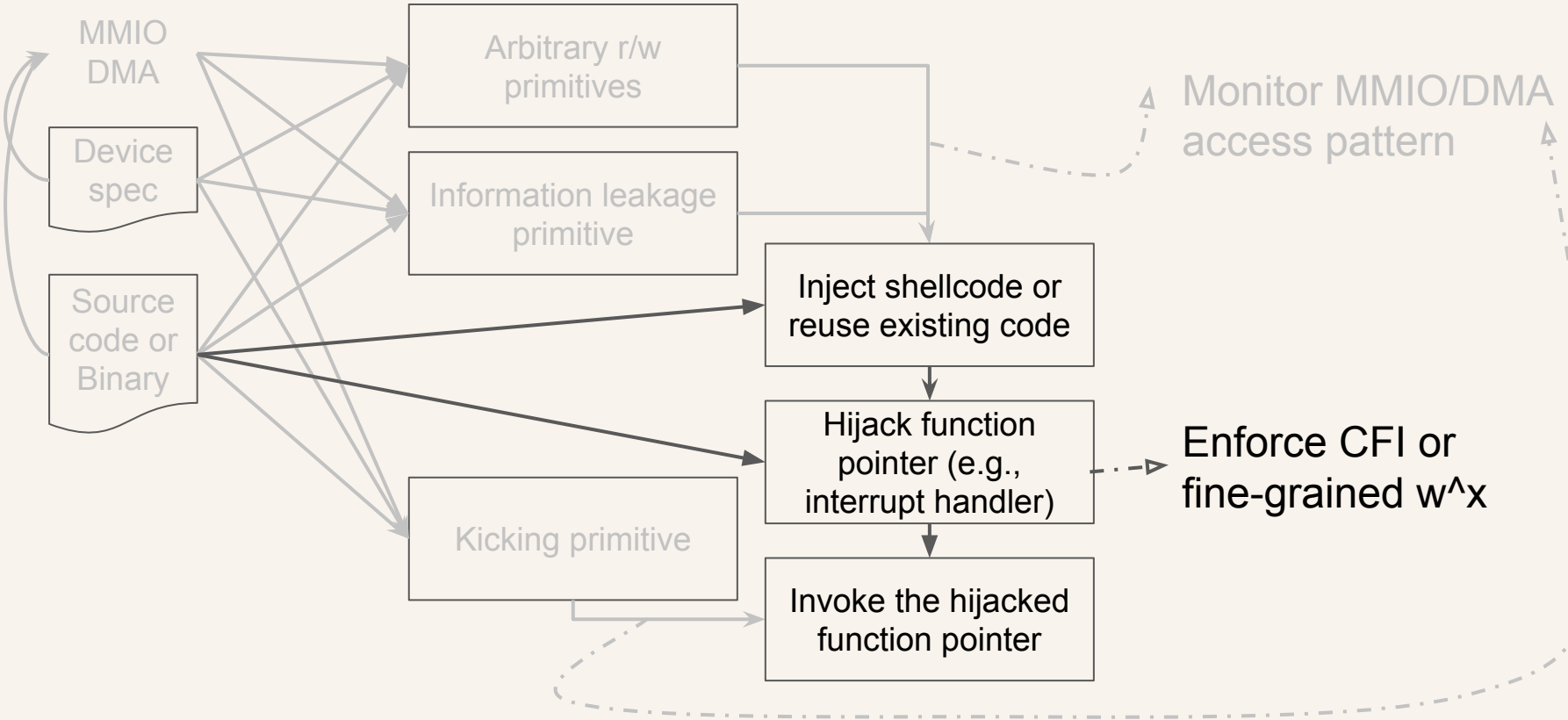
Email: qiang.liu@epfl.ch

Backup Slides

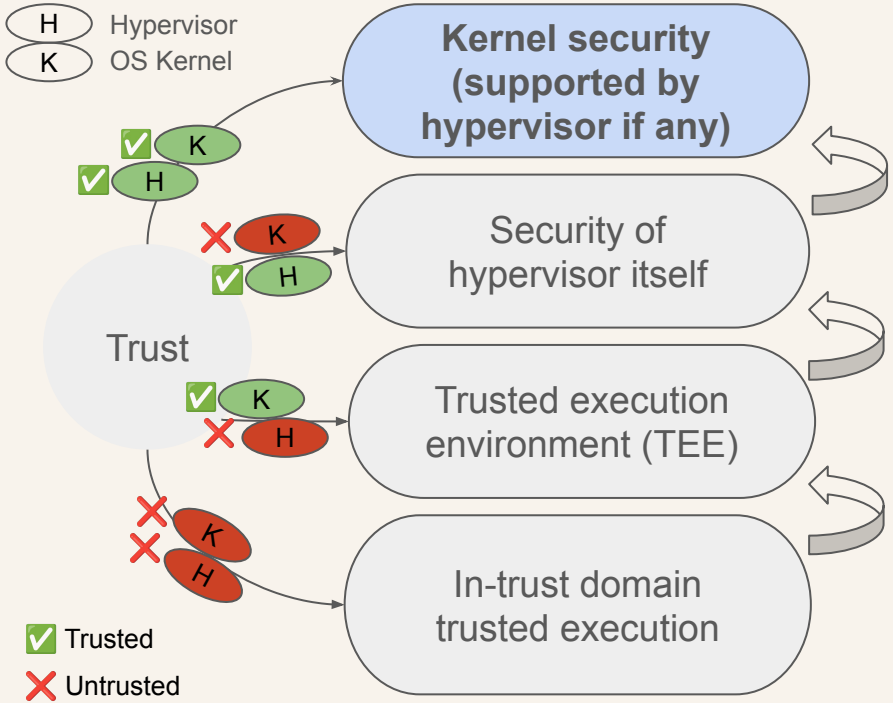
How to stop virtual device exploits in the wild?



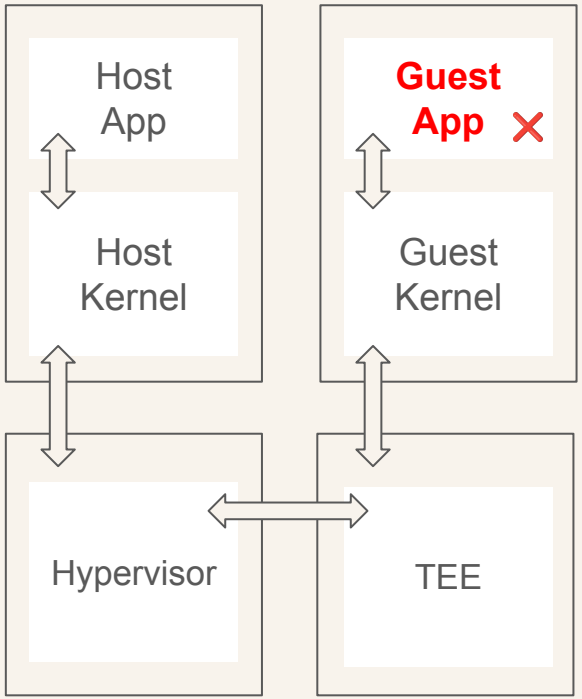
How to stop virtual device exploits in the wild?



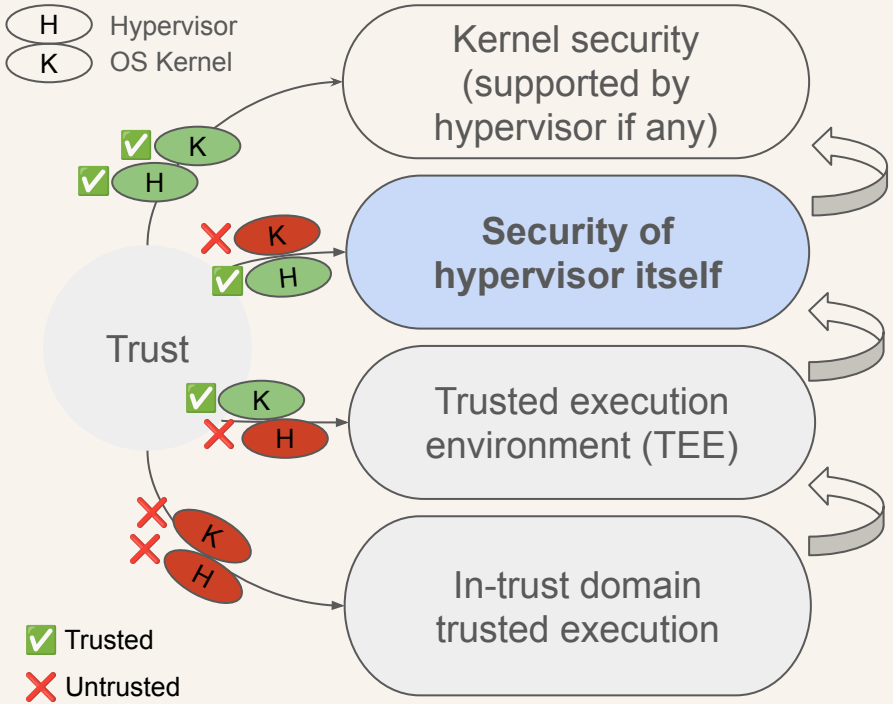
Explicit and transitive trust of the kernel and hypervisor



FirmGuide [ASE21]
ECMO [CCS21]

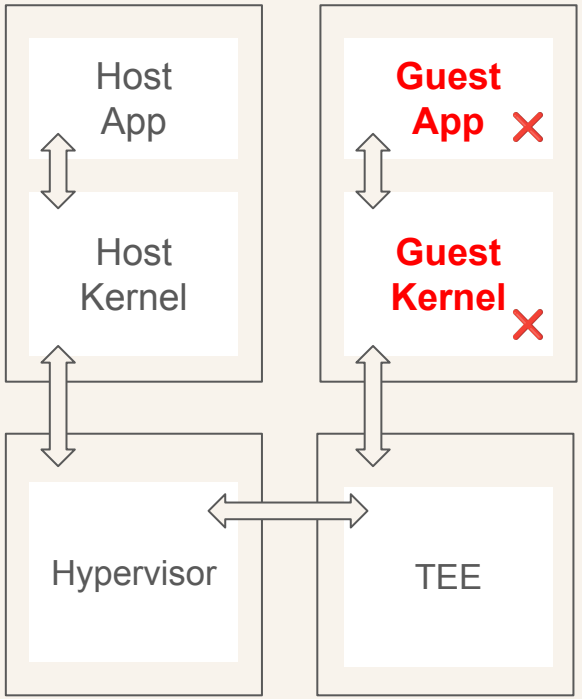


Explicit and transitive trust of the kernel and hypervisor

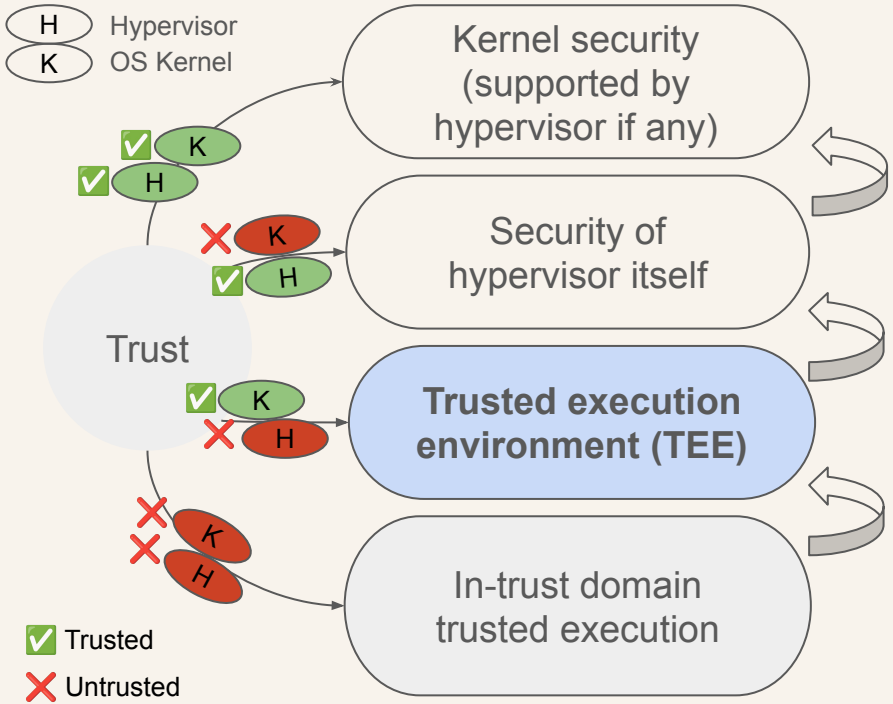


FirmGuide [ASE21]
ECMO [CCS21]

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

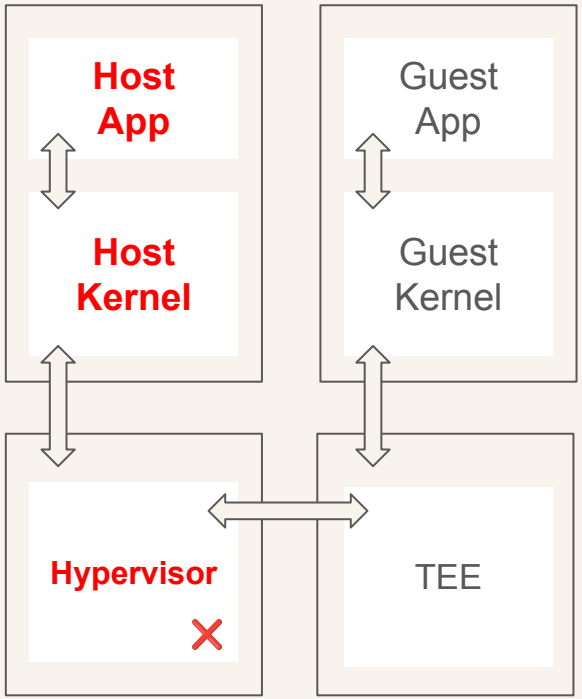


Explicit and transitive trust of the kernel and hypervisor

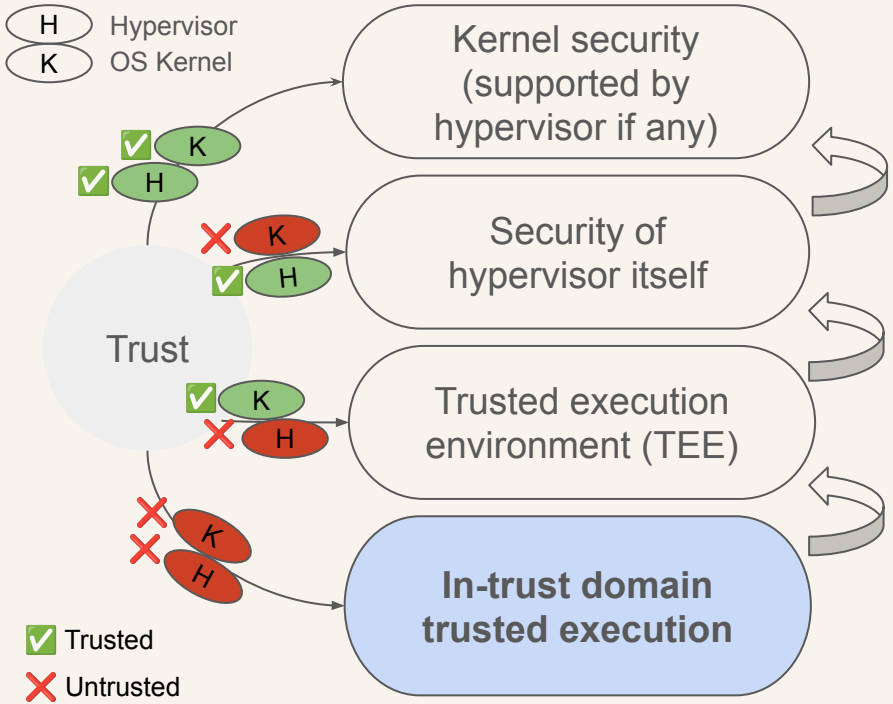


FirmGuide [ASE21]
ECMO [CCS21]

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

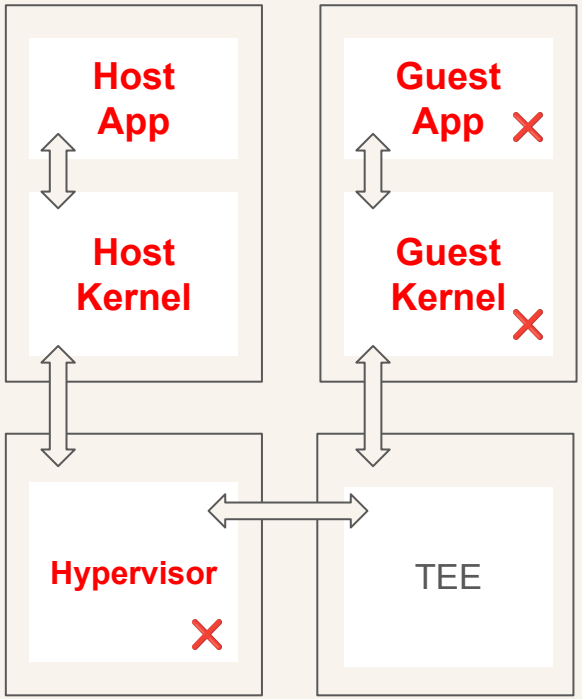


Explicit and transitive trust of the kernel and hypervisor

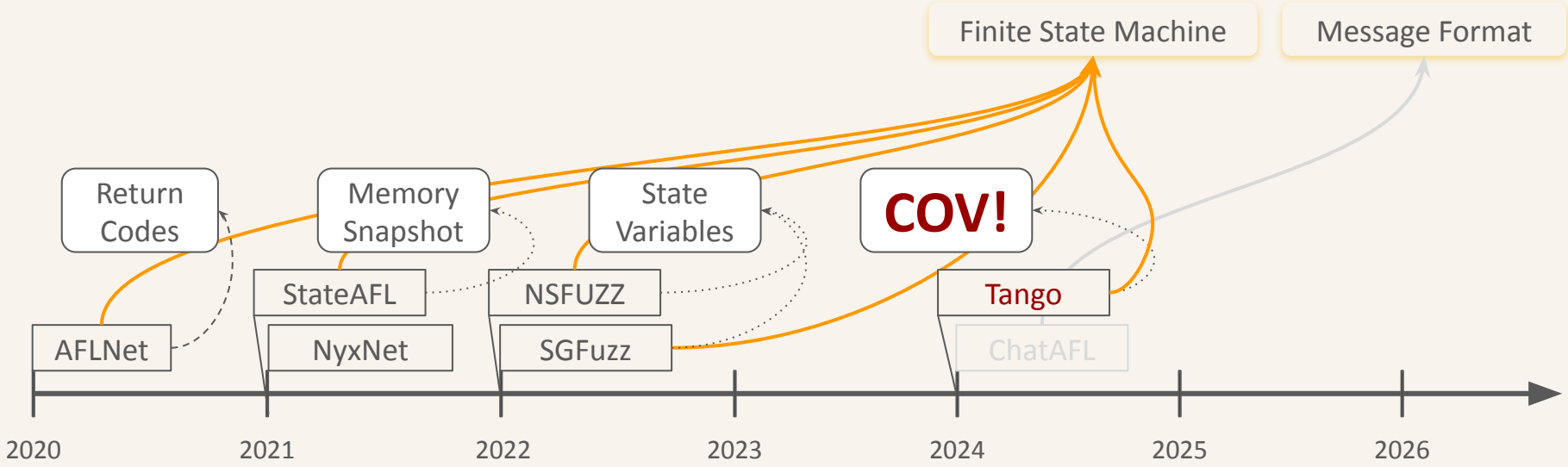


FirmGuide [ASE21]
ECMO [CCS21]

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?

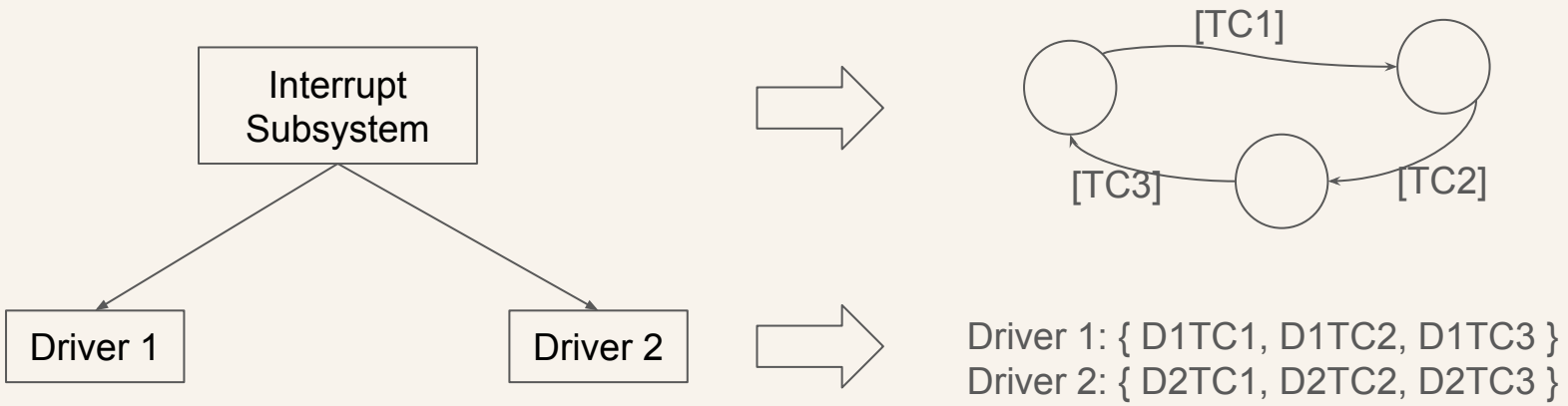
- Challenges: ARM/MIPS devices have fragmented peripherals
- Aim at a minimum best effort to boot with an interactive shell [FirmGuide ASE21]

ARM chip example: plxtech,nas782x			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

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]

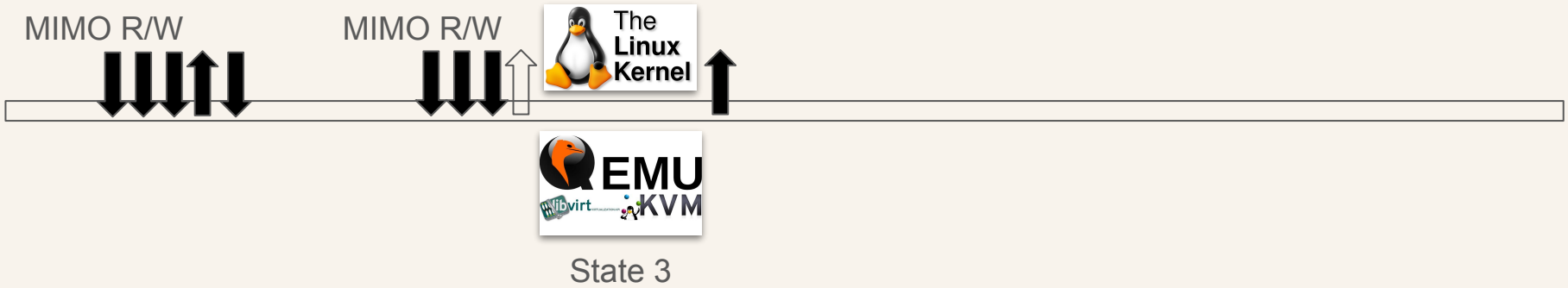
Linux kernel runs

Model-guided kernel execution FirmGuide [ASE21]

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

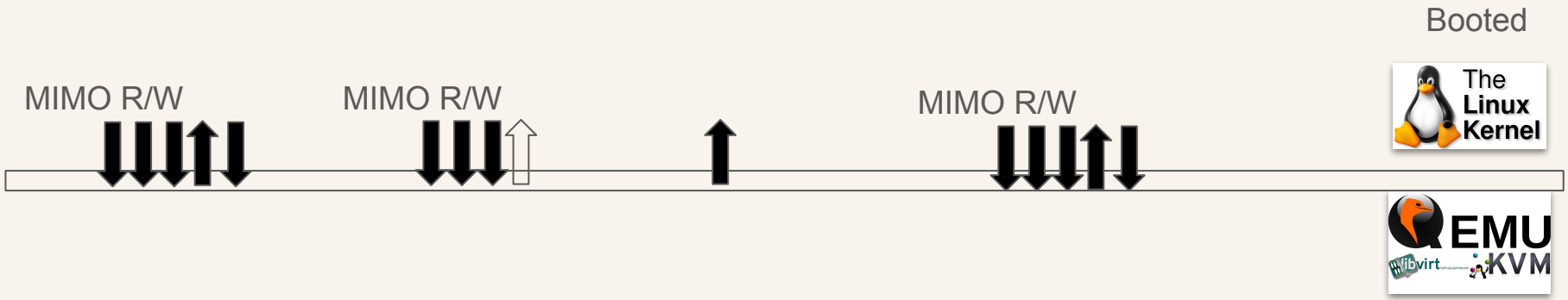
Model-guided kernel execution FirmGuide [ASE21]

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.