Core Slicing: Closing the gap between leaky confidential VMs and bare-metal cloud

Ziqiao Zhou, Yizhou Shan, Weidong Cui, Xinyang Ge, Marcus Peinado, Andrew Baumann
**Goal**: Remove hypervisor from TCB

**Solution**: Deprivilege hypervisor

**Examples**: AMD SEV, Intel TDX, Arm CCA
Background: Confidential VMs

- **Goal**: Remove hypervisor from TCB
- **Solution**: Deprivilege hypervisor
- **Examples**:  
  - AMD SEV, Intel TDX, Arm CCA
Background: Confidential VMs

- **Goal**: Remove hypervisor from TCB
- **Solution**: Deprivilege hypervisor
- **Examples**:
  - AMD SEV, Intel TDX, Arm CCA

Hypervisor still runs in the same core with VMs
Never-ending side channels
Never-ending side channels

Execution-unit contention (SP’23)
Never-ending side channels

<table>
<thead>
<tr>
<th>Transient execution attacks</th>
<th>CVE</th>
<th>Intel</th>
<th>AMD</th>
<th>ARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meltdown/Supervisor-only bypass</td>
<td>CVE-2017-5754</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Bound check bypass (Spectre-1)</td>
<td>CVE-2017-5753</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Branch Target Injection (Spectre-2)</td>
<td>CVE-2017-5715</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Speculative Store Bypass (Spectre-NG-4)</td>
<td>CVE-2018-3639</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Rogue System Register Read (Spectre-NG-3a)</td>
<td>CVE-2018-3640</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Lazy FP State Restore (Spectre-NG)</td>
<td>CVE-2018-3665</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>ForeShadow</td>
<td>CVE-2018-3615</td>
<td>Y</td>
<td>M</td>
<td>U</td>
</tr>
<tr>
<td>Bounds Check Bypass Store</td>
<td>CVE-2018-3693</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Straight-line Speculation</td>
<td>CVE-2021-26341</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Return Stack Buffer (Spectre-RSB)</td>
<td>CVE-2022-29901\nCVE-2022-23824</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Speculative Vectorization Exploits (Spectre-HD) (2023-02)</td>
<td></td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

...
Confidential VMs: Reactive mitigations
Confidential VMs: **Reactive** mitigations
Confidential VMs: Reactive mitigations

“Intel believes removing all incidental channels from computing systems is not in customer’s best interests and is not feasible, nor is it feasible to completely prevent the intentional misuse of incidental channels.”

-Intel
How to eliminate side-channel attacks

- Adversary-controlled code in the same core
How to eliminate side-channel attacks

- Adversary-controlled code in the same core
Existing solution: bare-metal cloud
Existing solution: bare-metal cloud

• **Pros**: Strong isolation, predicable performance
• **Cons**: Lack of flexibility
Existing solution: bare-metal cloud

- **Pros**: Strong isolation, predicable performance
- **Cons**: Lack of flexibility

We want “bare-metal” security and performance, but at sub-machine granularity.
Hypervisor in today’s IaaS cloud

- VM 2 cores
- VM 1c
- host OS
- Hypervisor
- core
- core
- core
- core
- cache
- DRAM
Hypervisor in today’s IaaS cloud

**Resources sold should match those available.**

- Discrete cores
  - No time slicing
- Static memory
  - No ballooning or demand paging
- I/O offload
Hypervisor in today’s IaaS cloud

**Resources sold should match those available.**

- Discrete cores
  - No time slicing
- Static memory
  - No ballooning or demand paging
- I/O offload

<table>
<thead>
<tr>
<th>Core</th>
<th>Core</th>
<th>Core</th>
<th>Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM 2 cores</td>
<td>VM 1c</td>
<td>host OS</td>
<td></td>
</tr>
</tbody>
</table>

**Diagram:**

- Hypervisor
- Core
- Cache
- DRAM
Hypervisor in today’s IaaS cloud

**Resources sold should match those available.**

- Discrete cores
  - No time slicing
- Static memory
  - No ballooning or demand paging
- I/O offload
Hypervisor in today’s IaaS cloud

Resources sold should match those available.

- Discrete cores
  - No time slicing
- Static memory
  - No ballooning or demand paging
- I/O offload

Hypervisor is only used for isolation.
Hypervisor in today’s IaaS cloud

Resources sold should match those available.

- Discrete cores
  - No time slicing
- Static memory
  - No ballooning or demand paging
- I/O offload

Hypervisor is the source of CPU sharing.

Hypervisor is only used for isolation.
Hypervisor in today’s IaaS cloud

Resources sold should match those available.

• Discrete cores
  • No time slicing
• Static memory
  • No ballooning or demand paging
• I/O offload

Hypervisor is the source of CPU sharing.

Hypervisor is only used for isolation.
Idea of core slicing

✔ Each slice gets a physical partition
  • Exclusive CPU
    • No CPU virtualization layer
  • Exclusive DRAM partition
    • No additional memory translation
  • Directly access dedicated I/O devices
    • e.g., access virtual devices via SR-IOV
Idea of core slicing

✓ Each slice gets a physical partition
  • Exclusive CPU
    • No CPU virtualization layer
  • Exclusive DRAM partition
    • No additional memory translation
  • Directly access dedicated I/O devices
    • e.g., access virtual devices via SR-IOV
Idea of core slicing

✓ Each slice gets a physical partition
  • Exclusive CPU
    • No CPU virtualization layer
  • Exclusive DRAM partition
    • No additional memory translation
  • Directly access dedicated I/O devices
    • e.g., access virtual devices via SR-IOV

✓ Hardware-assisted isolation
Two lightweight hardware features

• **Per-core lockable filter** registers
  • Defines hardware resources (e.g., DRAM)
  • Locked until a secure reset

• A secure **per-core reset** unit
  • Clear per-core state
Two lightweight hardware features

• **Per-core lockable filter** registers
  • Defines hardware resources (e.g., DRAM)
  • Locked until a secure reset

• A secure **per-core reset** unit
  • Clear per-core state
Two lightweight hardware features

- **Per-core lockable filter** registers
  - Defines hardware resources (e.g., DRAM)
  - Locked until a secure reset

- A secure **per-core reset** unit
  - Clear per-core state
Lockable filter registers: Slicing DRAM

• Per-core lockable filter registers
  • Allowlist and denylist for DRAM ranges
Lockable filter registers: Slicing DRAM

• Per-core lockable filter registers
  • Allowlist and denylist for DRAM ranges
Lockable filter registers: Slicing DRAM

• Per-core lockable filter registers
  • Allowlist and denylist for DRAM ranges
Lockable filter registers: Slicing DRAM

- Per-core lockable filter registers
  - Allowlist and denylist for DRAM ranges
Lockable filter registers: Slicing DRAM

• Per-core lockable filter registers
  • Allowlist and denylist for DRAM ranges
Lockable filter registers: Slicing DRAM

• Per-core lockable filter registers
  • Allowlist and denylist for DRAM ranges

![Diagram showing lockable filter registers and their interactions with SW Lock, SW Write, and Per-core reset.]

Slice A
Core

Slice B
Core

DRAM
Lockable filter registers: Slicing DRAM

- Per-core lockable filter registers
  - Allowlist and denylist for DRAM ranges
Lockable filter registers: Slicing DRAM

• Per-core lockable filter registers
  • Allowlist and denylist for DRAM ranges
Lockable filter registers: Slicing DRAM

- Per-core lockable filter registers
  - Allowlist and denylist for DRAM ranges
Slicing other resources

• Interrupts
• I/O devices
• Cache
  • SiFive: way masking for the shared cache.
  • Intel: Cache Allocation Technology (CAT).
• DMA
  • RISC-V: IOPMP
  • X86: IOMMU
Core slicing software stack

Host slice

Core
DRAM
I/O devices

slice $U_k$

Core
DRAM
I/O devices

bootROM
Core slicing software stack

- Bootloader (sliceloader) is **trusted**
- Slicevisor is **trusted** only at **control** path
  - Slice creation
  - Slice destroy.
- Untrusted firmware in guest slices
Core slicing software stack

- Bootloader (sliceloader) is **trusted**
- Slicevisor is **trusted** only at control path
  - Slice creation
  - Slice destroy.
- Untrusted firmware in guest slices

Read a protected slice configuration to determine and validate the role of the core
Core slicing software stack

- Bootloader (sliceloader) is **trusted**
- Slicevisor is trusted only at control path
  - Slice creation
  - Slice destroy.
- Untrusted firmware in guest slices
Core slicing software stack

- Bootloader (sliceloader) is trusted
- Slicevisor is trusted only at control path
  - Slice creation
  - Slice destroy.
- Untrusted firmware in guest slices
Core slicing software stack

- Bootloader (sliceloader) is trusted
- Slicevisor is trusted only at control path
  - Slice creation
  - Slice destroy.
- Untrusted firmware in guest slices
Core slicing software stack

- Bootloader (sliceloader) is **trusted**
- Slicevisor is **trusted** only at control path
  - Slice creation
  - Slice destroy.
- Untrusted firmware in guest slices

No trusted and shared software cross slices
Our prototype in RISC-V

PMP for
- DRAM isolation
- Memory-mapped
  - MSRs
  - IO
  - interrupts
Our prototype in RISC-V

PMP for
• DRAM isolation
• Memory-mapped
  • MSRs
  • IO
• interrupts

```
verification success.
hart_count = 2
mem_size = 20000000
digest:
0c08e3f644174b86e10284fca26aba368b79d89404342c9f80b135daa829a7616e546357
```

```bash
>> slice help
[72.740933] slice_help(): slice STOP -- stop a slice.
[72.744336] slice_help(): slice START -- start a slice.
[72.745143] slice_help(): slice CREATE -- create a slice.
[72.745744] slice_help(): slice DELETE -- delete a slice.
[72.746607] slice_help(): slice ATTEST -- attest a slice.
```

```
Jan 1 00:00:02 login[83]: root login on 'con'
~ # more /proc/cpuinfo
processor : 0
hart : 1
isa : rv64imafdc
mmu : sv39
uarch : sifive,rocket0
```

```
Jan 1 00:00:05 login[90]: root login on 'con'
~ # cat /proc/cpuinfo
processor : 0
hart : 3
isa : rv64imafdc
mmu : sv39
uarch : sifive,rocket0
```
Our prototype in RISC-V

PMP for
- DRAM isolation
- Memory-mapped
  - MSRs
  - IO
  - interrupts

Exclusive resources
Our prototype in RISC-V

PMP for
• DRAM isolation
• Memory-mapped
  • MSRs
• IO
• interrupts

Exclusive resources
Bare-metal performance
Summary

• TEE that avoids side channels by design
  • No virtualization overhead
  • Resources partitioned at core granularity

• Two lightweight hardware features
  • Lockable filter registers
  • Per-core reset

• More details in the paper, including:
  • Attestation and memory encryption
  • Extending the design beyond RISC-V
  • Evaluation of limited registers and bare-metal performance

Artifact available at https://github.com/msrssp/core-slicing