KVM Address Space Isolation (ASI)

Alexandre Chartre – Oracle
Ofir Weisse, Junaid Shahid, Oleg Rombakh, and Paul Turner – Google
KVM Summit – October 2020
Safe harbor statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle’s products may change and remains at the sole discretion of Oracle Corporation.
Why ASI?
Context

- Data can leak between CPU threads from the same CPU core
  - Leak through shared hardware (micro)architecture via speculative attacks
  - Example: L1TF and MDS speculative attacks
Context

- Data can leak between CPU threads from the same CPU core
  - Leak through shared hardware (micro)architecture via speculative attacks
  - Example: L1TF and MDS speculative attacks

- A VM can control the leak and spy on its sibling CPU thread
  - Guest can spy on another guest running on the same CPU core
  - Guest can spy on the host running on the same CPU core

- Major issue for virtual machines and cloud providers
  - Allow Guest-to-Guest attacks and Guest-to-Host attacks
Mitigations

- Basic mitigation: disable CPU hyper-threading
  - Most complete and reliable solution
  - Significant impact on performances and capacity

- Mitigation for Guest-to-Guest attacks
  - Pin VMs to different dedicated CPU cores
  - Core scheduling

- Mitigation for Guest-to-Host attacks
  - Synchronize VM entry and VM exit for all CPU threads on a CPU Core, i.e., halt sibling core when running host code
  - Flush L1D cache before every VMENTER - not cheap
  - KVM Address Space Isolation (ASI)
ASI Intuition - Can’t Speculate Through a Page Fault

Trivial example: Spectre V1 (bounds check bypass)

```c
int foo(u8 *arr, int size, int index) {
    if (index < size) {
        // Should I fence
        return global_array[ arr[index] * 64 ];
    }
    // ...
}
```

If index is out of bounds, “arr” might speculatively still be accessed. If &arr[index] is not mapped in the page-table → page-fault
ASI Overview

- Define a restricted address space
  - Define a page table with limited data
  - Contain no sensitive or secret data
- Prevent a sub-system from accessing the entire memory or unrelated data
- Sub-system explicitly enters/exits ASI
- ASI can be interrupted/resumed on interrupt, exception, context switch
- ASI can be suspended/exited on page-fault
- **Accessing secrets → page fault**
ASI Applications

- **KVM ASI**
  - Protect against guest-to-host attack
  - Major challenge: what data to include in the ASI domain?

- **User ASI**
  - Implement kernel/user page-table switch with ASI
  - Refactor Kernel Page-Table Isolation (KPTI) to use ASI

- **Userland ASI?**
  - Provide multiple user address spaces to a user process
  - Isolate user virtual environment (JVM, containers…)
  - To be investigated
KVM ASI
KVM ASI

- Address space with limited kernel and VM mappings
  - Only has mappings required to enter VM and handle VM exit

- Goal: run VM and handle (most) VM exits without exiting ASI

- Only map data from a single VM in the same ASI
  - Prevent VM running on same CPU core to steal data from host kernel or from another VM

- Synchronize on VM entry only if sibling CPU thread is not running ASI
  - No need to synchronize if all CPU threads are running with ASI
  - Core scheduling helps having the same VM ASI run on all CPU threads
ASI Lifecycle

- Create an ASI
  - Each ASI has its own page-table
- Populate the ASI page-table
- Enter ASI = switch to the ASI page-table
- Handle interrupt/exception/fault/context switch
  - ASI can be interrupted and resumed, or exited
- Exit ASI = switch to the kernel page-table
- Destroy the ASI page-table
KVM ASI Usage

- Create one KVM ASI per VM (or per VCPU)
- Populate the KVM ASI page-table
- Use KVM ASI when running a guest VCPU
  - KVM_RUN ioctl
  - Enter KVM ASI
  - Ensure KVM ASI is used on siblings
    - VMEnter
    - VMExit
    - KVM ASI also used on siblings
  - Stop KVM ASI enforcement on siblings
  - Run (most) VMExit handlers with KVM ASI
  - vcpu_run() loop
ASI Page-Table Filling

- Basic solution: manually add each required mapping
  - asi_map(asi, address, length);
  - asi_unmap(asi, address);
- Similar to memory allocation/free
- Works okay for pre-allocated well-known buffers
- More cumbersome for frequently re-allocated buffers
- Need mechanisms for adding mappings more automatically
ASI Page-Table Filling - Statically Allocated Buffers

- Put non-sensitive statically allocated data into a dedicated object file section
- Just require to add a compiler section attribute to the data definition

```
#define __asi_not_sensitive __attribute__((__section__(ASI_NON_SENSITIVE_SECTION_NAME)))

static struct clocksource clocksource_tsc __asi_not_sensitive = {
    .name = "tsc",
    .rating = 300,
    .read = read_tsc,
```

- Then map the entire section into the ASI
ASI Page-Table Filling - Dynamically Allocated Buffers

- Use new flag when allocating a non-sensitive buffer:
  - GFP_GLOBAL_NONSENSITIVE - non-sensitive data for any ASI
    - GFP_LOCAL_NONSENSITIVE - non-sensitive data for the local process

- Buffer is automatically [un]mapped into the current ASI on alloc/free
ASI Page-Table Switching

- Simple on x86: just update the CR3 control register
  - But inefficient if you don’t consider TLB

- TLB = Translation Lookaside Buffer
  - Caches VA-to-PA translations, avoid MMU to go through the page-table
  - Switching require flushing the TLB → impact on performances

- Optimization: use Process-Context Identifiers (PCIDs)
  - Facility to associate a TLB entry with a page-table
  - Avoid flushing the entire TLB
  - Flush only TLB entries for a specified PCID
ASI and Interrupts/Exceptions

- Should be able to handle interrupts/exceptions while running with ASI
  - But ASI doesn’t necessarily have all mappings to run the interrupt/exception handler

- Suspend/Resume ASI while running interrupt/exception handler
  - Exit ASI at the beginning of the handler
  - Re-enter ASI at the end of the handler
  - Solution when we know the handler needs the kernel page-table
ASI and Interrupts/Exceptions

- Run the interrupt/exception handler with the ASI page-table
- Handler can fault because of the ASI restricted page-table
- Page-fault handler will then exit ASI and resume handler execution
- ASI is re-entered after the handler is done (if there was a fault)
- Solution when we expect the handler to be able to run with ASI
ASI and Interrupts/Exceptions

Interrupt Handler

ASI

Page-Fault Handler

kernel page-table

Page-fault

interrupt

Code

ASI

Intr. Handler (cont.)

kernel page-table

page-fault return

interrupt return

Code (cont.)

ASI

Intr. Handler (cont.)
ASI and Page Fault

- Basic behavior:
  - Exit ASI and retry with kernel page-table
  - Log fault to identify missing mapping in ASI page-table
  - Eventually manually augment the ASI page-table to prevent this fault

- Can we do better?
  - Automatically add missing mapping to the ASI page-table
  - Only if mapping does not provide access to sensitive data
  - Allow to return with ASI and avoid the same fault to happen again
  - Require to flag kernel memory which has no sensitive data (e.g. GFP_GLOBAL_NONSENSITIVE)
ASI and Context Switch

- Task using ASI is scheduled out → interrupt ASI
  - Exit ASI and save ASI information for this task

- Task using ASI is scheduled in → resume ASI
  - Enter ASI with the saved ASI information for this task

- Additional complexity if switch occurs during interrupt/exception handler
  - Interrupt/exception might already have exited ASI
  - Need to save/restore ASI state information
ASI Synchronization Across CPU Threads

- Mechanism to force all CPU threads from a CPU core to use a specified ASI

- If sibling CPU thread is running ASI
  - CPU thread continues to run uninterrupted
  - If CPU thread tries to exit ASI when it waits in idle loop

- If sibling CPU thread is not running ASI
  - CPU thread is requested to reschedule
  - If next task is using ASI then enter ASI and run the task
  - If next task is not using ASI then enter ASI and wait in idle loop

- KVM ASI uses ASI synchronization while running a VM
  - Synchronization is started before VMEnter
  - Synchronization is stopped after VMExit
KVM ASI Synchronization with Siblings Running ASI

Sibling CPU 1

Sibling CPU 2

CPU running KVM

peek for ASI

peek for ASI

ASI exit

wait before exiting

start synchronization

stop synchronization

ASI idle loop

Kernel

ASI

VM

ASI
KVM ASI Synchronization with Siblings not Running ASI

Sibling CPU 1

Kernel → ASI

peek for ASI → send reschedule()

next task is using ASI
run task with ASI

Sibling CPU 2

Kernel → ASI idle loop → Kernel

next task is not using ASI
switch to ASI and wait

CPU running KVM

ASI → VM → ASI

peek for ASI → send reschedule()

start synchronization
stop synchronization
ASI Synchronization and Interrupt/Exception

- If Interrupt/Exception needs to exit ASI
- Then this will cause all sibling CPU threads to interrupt
- One CPU thread receives an interrupt/exception, touches secret data → PF → ASI-exit
- That CPU thread forces all sibling CPU threads to interrupt
  - All CPU threads exit ASI, wait for interrupt to be processed, return to ASI
  - Behavior is synchronized across all CPU threads
- Core Scheduling has a somewhat similar mechanism
  - To prevent interrupt/exception handler to run with tagged process
ASI Page Tables
The KPTI Model - Control & Data Privilege

Userspace page-table

<table>
<thead>
<tr>
<th>Userspace</th>
</tr>
</thead>
<tbody>
<tr>
<td>userspace</td>
</tr>
</tbody>
</table>

Kernel page-table

<table>
<thead>
<tr>
<th>Kernel page-table</th>
</tr>
</thead>
<tbody>
<tr>
<td>userspace</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Global kernel data</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Process A data</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Process B data</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Other stuff we'll ignore for now</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Kernel text, modules, globals</td>
</tr>
</tbody>
</table>

To mitigate Meltdown attacks, KPTI differentiates between privileged/unprivileged execution level.

The methodology - using two page tables to separate between user space memory and kernel privileged memory.

Direct map via kmalloc

We'll ignore vmalloc space for now.
It is conceptually similar to direct map

We'll also ignore global vars
In ASI, we define privilege based on data access, not execution-level. We add another “restricted” page-table which only maps kernel non-sensitive data.

Data is deemed non-sensitive if, when stolen by a malicious VM, does not pose a security threat to other VMs or cloud infrastructure.

For performance reasons, we’re interested in memory that is accessed frequently by the kernel, when operating a VM between VMEXIT and VMENTER.
Non-sensitive data can be accessed freely, without the need for any L1TF mitigations.

Access to “unmapped” area will cause a PF, which will switch to the unrestricted page-table. Use L1TF mitigation when switching (stunning/L1D-flush).
Is data considered non-sensitive **locally** in a process or **globally** in the entire system?

Examples:
1. **Local data:** VMCS, vcpu, file-descriptor-table
2. **Global data:** sk_buffs

All non-sensitive data in ASI can be read by a guest VM via an L1TF attack

While we want VM-1 to access its VMCS freely, we don’t want VM-1 to read the VMCS of VM-2!!
Early Results
## Initial Results - Aeropspike YCSB

### Ratio of ASI-exits/VM-exits

<table>
<thead>
<tr>
<th>KVM/VCPU</th>
<th>0xffffc9003f575000/0: Time 139.64 seconds, asi/vm exits</th>
<th>2504149 / 4816824 = 51.99 %</th>
</tr>
</thead>
<tbody>
<tr>
<td>KVM/VCPU</td>
<td>0xffffc9003f575000/1: Time 139.51 seconds, asi/vm exits</td>
<td>355404 / 657455 = 54.06 %</td>
</tr>
<tr>
<td>KVM/VCPU</td>
<td>0xffffc9003f575000/2: Time 139.51 seconds, asi/vm exits</td>
<td>121572 / 481107 = 25.27 %</td>
</tr>
<tr>
<td>KVM/VCPU</td>
<td>0xffffc9003f575000/3: Time 139.51 seconds, asi/vm exits</td>
<td>214304 / 439711 = 48.74 %</td>
</tr>
<tr>
<td>KVM/VCPU</td>
<td>0xffffc9003f575000/4: Time 139.51 seconds, asi/vm exits</td>
<td>136400 / 351173 = 38.84 %</td>
</tr>
<tr>
<td>KVM/VCPU</td>
<td>0xffffc9003f575000/5: Time 139.51 seconds, asi/vm exits</td>
<td>163717 / 445313 = 36.76 %</td>
</tr>
<tr>
<td>KVM/VCPU</td>
<td>0xffffc9003f575000/6: Time 139.51 seconds, asi/vm exits</td>
<td>110505 / 360235 = 30.68 %</td>
</tr>
<tr>
<td>KVM/VCPU</td>
<td>0xffffc9003f575000/7: Time 139.51 seconds, asi/vm exits</td>
<td>135873 / 393257 = 34.55 %</td>
</tr>
</tbody>
</table>

Total asi exits = 3741924

<table>
<thead>
<tr>
<th>KVM/VCPU</th>
<th>0xffffc90019e44000/0: Time 233.39 seconds, asi/vm exits</th>
<th>2524400 / 4920423 = 51.30 %</th>
</tr>
</thead>
<tbody>
<tr>
<td>KVM/VCPU</td>
<td>0xffffc90019e44000/1: Time 233.26 seconds, asi/vm exits</td>
<td>368409 / 686433 = 53.67 %</td>
</tr>
<tr>
<td>KVM/VCPU</td>
<td>0xffffc90019e44000/2: Time 233.26 seconds, asi/vm exits</td>
<td>114840 / 392239 = 29.28 %</td>
</tr>
<tr>
<td>KVM/VCPU</td>
<td>0xffffc90019e44000/3: Time 233.26 seconds, asi/vm exits</td>
<td>194565 / 461824 = 42.13 %</td>
</tr>
<tr>
<td>KVM/VCPU</td>
<td>0xffffc90019e44000/4: Time 233.26 seconds, asi/vm exits</td>
<td>108116 / 384302 = 28.13 %</td>
</tr>
<tr>
<td>KVM/VCPU</td>
<td>0xffffc90019e44000/5: Time 233.26 seconds, asi/vm exits</td>
<td>163987 / 457225 = 35.87 %</td>
</tr>
<tr>
<td>KVM/VCPU</td>
<td>0xffffc90019e44000/6: Time 233.26 seconds, asi/vm exits</td>
<td>100777 / 460626 = 21.88 %</td>
</tr>
<tr>
<td>KVM/VCPU</td>
<td>0xffffc90019e44000/7: Time 214.20 seconds, asi/vm exits</td>
<td>90452 / 335486 = 26.96 %</td>
</tr>
</tbody>
</table>

Total asi exits = 3665546
### Initial Results - Aerospoke YCSB

#### Exit details

<table>
<thead>
<tr>
<th>RIP</th>
<th>address</th>
<th>accessor</th>
<th>allocator</th>
<th>count</th>
<th>CDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0xfffffffff81e33b9</td>
<td>0xffffffffc9001e55ca8</td>
<td>kernel/rcu/srcutree.c:410</td>
<td>../arch/x86/kvm/vmx.c:11916</td>
<td>80677</td>
<td>1.000000</td>
</tr>
<tr>
<td>1 0xfffffffff81e77d</td>
<td>0xfffffe8ffe3c8730</td>
<td>../lib/timerqueue.c:52</td>
<td>../kernel/events/core.c:10523</td>
<td>51266</td>
<td>0.926417</td>
</tr>
<tr>
<td>2 0xfffffffff81e77d</td>
<td>0xfffffe8ffe348730</td>
<td>../lib/timerqueue.c:52</td>
<td>../kernel/events/core.c:10523</td>
<td>45283</td>
<td>0.879659</td>
</tr>
<tr>
<td>3 0xfffffffff81e77d</td>
<td>0xfffffe8ffe408730</td>
<td>../lib/timerqueue.c:52</td>
<td>../kernel/events/core.c:10523</td>
<td>31750</td>
<td>0.838358</td>
</tr>
<tr>
<td>4 0xfffffffff81e77d</td>
<td>0xffffffff8b6e49a7a0</td>
<td>../lib/timerqueue.c:52</td>
<td>../kernel/events/core.c:10523</td>
<td>31113</td>
<td>0.809394</td>
</tr>
<tr>
<td>5 0xfffffffff81e33b9</td>
<td>0xffffffffc9001f3b3ca8</td>
<td>kernel/rcu/srcutree.c:410</td>
<td>../arch/x86/kvm/vmx.c:11916</td>
<td>30228</td>
<td>0.781017</td>
</tr>
<tr>
<td>6 0xfffffffff81e77d</td>
<td>0xffffffff8ffe388730</td>
<td>../lib/timerqueue.c:52</td>
<td>../kernel/events/core.c:10523</td>
<td>28256</td>
<td>0.753447</td>
</tr>
<tr>
<td>7 0xfffffffff81e77d</td>
<td>0xffffffff8e699ddde8</td>
<td>../lib/timerqueue.c:52</td>
<td>&lt;Unknown alloc&gt;</td>
<td>27816</td>
<td>0.727675</td>
</tr>
<tr>
<td>8 0xfffffffff81e77d</td>
<td>0xffffffff8a000148730</td>
<td>../lib/timerqueue.c:52</td>
<td>../kernel/events/core.c:10523</td>
<td>23678</td>
<td>0.702305</td>
</tr>
<tr>
<td>9 0xfffffffff81e77d</td>
<td>0xffffffff8a0002c8730</td>
<td>../lib/timerqueue.c:52</td>
<td>../kernel/events/core.c:10523</td>
<td>20716</td>
<td>0.680709</td>
</tr>
<tr>
<td>10 0xfffffffff81e77d</td>
<td>0xffffffff8f8ffdc8730</td>
<td>../lib/timerqueue.c:52</td>
<td>../kernel/events/core.c:10523</td>
<td>15481</td>
<td>0.661815</td>
</tr>
</tbody>
</table>
## Initial Results - Aeropspike YCSB

### Exit details by allocation site

<table>
<thead>
<tr>
<th>Variable</th>
<th>count</th>
<th>CDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Unknown alloc&gt;</td>
<td>428458</td>
<td>1.000000</td>
</tr>
<tr>
<td>./kernel/events/core.c:10523</td>
<td>419826</td>
<td>0.609217</td>
</tr>
<tr>
<td>PO: ./arch/x86/kvm/vmx.c:11916</td>
<td>80677</td>
<td>0.226306</td>
</tr>
<tr>
<td>sd_llc_size</td>
<td>39383</td>
<td>0.152723</td>
</tr>
<tr>
<td>PO: ./arch/x86/kvm/vmx.c:11916</td>
<td>30228</td>
<td>0.088426</td>
</tr>
<tr>
<td>./arch/x86/kvm/.../virt/kvm/eventfd.c:1016</td>
<td>14221</td>
<td>0.060856</td>
</tr>
<tr>
<td>PO: ./fs/exec.c:1761</td>
<td>9620</td>
<td>0.047885</td>
</tr>
<tr>
<td>PO: ./kernel/fork.c:890</td>
<td>9263</td>
<td>0.039111</td>
</tr>
<tr>
<td>./fs/timerfd.c:391</td>
<td>6797</td>
<td>0.030663</td>
</tr>
<tr>
<td>PO: ./arch/x86/events/intel/uncore.c:321</td>
<td>4412</td>
<td>0.024464</td>
</tr>
<tr>
<td>../security/scudo/lsm.c:310</td>
<td>2896</td>
<td>0.020439</td>
</tr>
<tr>
<td>./arch/x86/kvm/.../virt/kvm/irqchip.c:212</td>
<td>2578</td>
<td>0.017798</td>
</tr>
<tr>
<td>PO: ./fs/binfmt_elf.c:466</td>
<td>2239</td>
<td>0.015447</td>
</tr>
<tr>
<td>intel_pmu_ops</td>
<td>1955</td>
<td>0.013405</td>
</tr>
<tr>
<td>PO: ./kernel/fork.c:1659</td>
<td>1639</td>
<td>0.011622</td>
</tr>
<tr>
<td>softnet_data</td>
<td>1519</td>
<td>0.010127</td>
</tr>
<tr>
<td>numa_node</td>
<td>1294</td>
<td>0.008741</td>
</tr>
</tbody>
</table>
ASI Status
History

- Idea suggested after L1TF speculative attack discovery
  - Initially introduced by Liran Alon in KVM Forum 2018 BoF
  - Inspired from Microsoft HyperV HyperClear L1TF mitigation
  - More discussions at Linux Plumbers Conference 2019

- Several RFCs submitted by Oracle
  - v1 (KVM ASI), v2 (Kernel ASI), v3 (+ integration with PTI), v4 (+ page-table management)
  - v5 (+ ASI synchronization)

- Different implementation recently proposed by Google
  - Presented at Linux Plumbers Conference 2020
Status and Future

- Collaboration between Oracle and Google
- Define a common solution and converge implementation
  - Page-table management
  - Interrupt/exception/page-fault handling
  - ASI synchronization across CPU threads (shared component with Core Scheduling?)
  - PTI integration
- Goal: upstream a common implementation
- Preliminary work: Defer PTI CR3 Switch to C Code
  - Simplify ASI support and integration with Page-Table Isolation (PTI)
Thank You

alexandre.chartre@oracle.com
oweisse@google.com