Address Space Isolation (ASI)

Alexandre Chartre
Oracle
Linux Plumbers Conference – August 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
- 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
  - KVM Address Space Isolation (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 Overview

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 is interrupted/resumed on interrupt, exception, context switch

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
- Interrupt/Resume ASI on interrupt/exception/fault/context switch
  - interrupt ASI = switch to the kernel page-table
  - resume ASI = switch back to the ASI page-table
- Exit ASI = switch to the kernel page-table
- Destroy the ASI page-table

ASI Applications

- KVM ASI
  - Protect against guest-to-host attack
- User ASI
  - Implement kernel/user page-table switch with ASI
  - Refactor KPTI to use ASI
- Userland ASI?
  - Provide multiple user address spaces to a user process
  - Isolate user virtual environment (JVM, containers…)
  - To be investigated

ASI and Page-Table

ASI Page Table

- An ASI page table (PGD) is allocated in a buffer at offset 0x1000
  - Used to identify ASI page-table vs kernel page-table
  - Mechanism already used for PTI page-tables
- Page-table is populated by adding VA range mappings
  - Copy mappings from the kernel page-table
  - Copy mapping entries at a specified level (e.g. PTE)
  - Can create cross-reference to the kernel page-table (if level != PTE)
- Need to keep track of pages used in the ASI page-table structure
  - To only modify/free entries effectively belonging to the ASI page-table

ASI VA Range Mappings

- Mapping VA range can leak adjacent data
  - Mapping size and alignment constrained by the page-table level (e.g. 4K at PTE level)
  - Mapping range can be larger than VA range and map (leak) adjacent data
  - Example: mapping a 8 bytes buffer at PTE level will leak at least one page (4K)
- Need to keep track of VA range mapped in the ASI page-table
  - To handle VA/mapping range overlap
  - Required for safely removing a mapping
  - Example: map different buffers from the same page => same mapping

Decorated Page Table (DPT)

- Encapsulate a native page-table (e.g. a PGD)
  - There is currently no generic function for creating/populating a page-table
  - Specific functions for mm, pti..
- Provide convenient page-table management functions
  - Track pages used in page-table structure
  - Track VA ranges mapped
  - Functions to add/remove VA range mapping
  - Handle VA ranges mapping overlap
- Used for building ASI page-table for KVM ASI and Test ASI
  - Could probably be used for PTI too

ASI Page-Table Switching

ASI Page Table Switching Logic

- On x86, just update the CR3 control register to switch the page-table
  - But also need to use Process-Context Identifiers (PCIDs) for efficiency
- PCIDs are used to limit TLB flushing on context (mm) switch
  - 6 PCIDs (0x001 to 0x006) are used to identify recently used mm kernel page-table
  - PCIDs are reused in a round-robin way for non-recently used mm
- Also used for user page-table (with PTI)
  - 6 PCIDs (0x801 to 0x806) are used to identify the corresponding mm user page-table
  - Switch between kernel and user → mask/unmask 0x800 in PCID

ASI and PCIDs

- Extend PCIDs usage to ASI as done with PTI
  - Change PTI implementation to use ASI (PTI → user ASI)
- Each ASI has a type with an associated PCID prefix (value between 1 and 255)
  - ASI PCID = (PCID prefix) << 4 | (kernel PCID)
  - User ASI → PCID prefix = 0x80 => user ASI PCID = [0x801, 0x806]
  - KVM ASI → PCID prefix = 0x01 => KVM ASI PCID = [0x011, 0x016]
- Track recently used ASI PCID (per ASI type)
  - Similar to mm context tracking for kernel PCID
  - Used to figure out if TLB flushing is required when switching to ASI page-table

ASI Page Table Switch Logic

- ASI Enter
  - Save the original CR3 value (CR3 value for the kernel page-table)
  - Get the kernel PCID used
  - Compute ASI PCID = (ASI PCID prefix << 4) | (kernel PCID)
  - Set flushing = FLUSH or NOFLUSH depending if ASI PCID is being reused or not
  - Update CR3 = flushing | PA(ASI page table) | ASI PCID
- ASI Exit
  - Update CR3 = saved original CR3 value | NOFLUSH

ASI, Interrupts and Friends

ASI and Interrupts/Exceptions

- Switch to kernel page-table to handle interrupts/exceptions
  - ASI Interrupt: logic similar to ASI Exit but keep track of the ASI being used
- Switch back to the ASI page-table after interrupt/exception has been handled
  - ASI Resume: logic similar to ASI Enter but return to the ASI used
- Track interrupt nesting depth
  - Interrupt/Resume ASI on the outermost interrupt
- ASI Enter/Exit runs with interrupts disabled
  - To prevent conflict with ASI Interrupt/Resume

ASI and Page Fault

- Should we return to the ASI after a page fault on ASI?
  - It depends on the ASI type
  - User ASI → yes, return to the ASI page-table
  - KVM ASI → no, return to the kernel page-table (ASI abort)
- Configurable option, specific to each ASI type
  - Either abort the ASI and ASI Resume won’t switch back to the ASI page-table
  - Or continue regular processing and ASI resume will switch back to the ASI page-table
- Optionally, log the fault (+ stack)
  - Useful for tracking missing mapping in ASI page-table

ASI and Context Switch

- Task using ASI is scheduled out → interrupt ASI
  - ASI Exit + save ASI information in task
- Task using ASI is scheduled in → resume ASI
  - ASI Enter with ASI information saved in task
- If context switch occurs in interrupt/exception
  - ASI has already been interrupted
  - Only need to save/restore ASI session information

ASI and Non-Maskable Interrupt (NMI)

- NMI can interrupt any code
  - In particular ASI Enter/Exit/Interrupt/Resume sequence
- Special processing on NMI Entry
  - Save CR3 value
  - Check if ASI is active based on CR3 value (ASI page table is at offset 0x1000)
  - If ASI is enabled then switch to the kernel page-table
- Special processing on NMI return
  - Check if ASI was active based on the saved CR3 value
  - If ASI was active then check if flushing is needed
  - Restore the saved CR3 value with flushing if needed

ASI Lockdown

ASI Lockdown

- Force all CPU threads from a CPU core to use a specified ASI
- If sibling CPU thread is already running the ASI
  - CPU thread continues to run uninterrupted
  - CPU thread will wait in idle loop if it tries to exit ASI
- If sibling CPU thread is not running the ASI
  - CPU thread is requested to reschedule
  - If next task is using the ASI then enter the ASI and run the task
  - If next task is not using the ASI then enter the ASI and wait in idle loop
- ASI lockdown tag allows different ASIs to be locked down together
  - Lockdown together ASIs which have the same lockdown tag
  - By default, each ASI has a unique lockdown tag

ASI Lockdown and Interrupt/Exception/NMI

- On ASI lockdown
  - Interrupt/Exception/NMI need to interrupt ASI
  - Interrupt/Exception/NMI will cause **all** sibling CPU threads to interrupt
- One CPU thread receives an interrupt/exception/NMI
- That CPU thread forces all sibling CPU threads to interrupt
- Then each CPU thread will:
  - Wait for all sibling CPU threads to be interrupted
  - Interrupt ASI → switch to kernel page-table
  - Do interrupt processing
  - Wait for interrupt processing to be done on all sibling CPU threads
  - Resume ASI → switch to ASI page-table
  - Resume from interrupt

ASI Lockdown and Fault

- Fault on ASI can cause ASI to exit
  - For example with KVM ASI
- If fault happens during ASI lockdown then lockdown is breached
  - Security mitigation might not be effective anymore
- Ideally we should then run single-threaded on the impacted CPU core
  - Certainly complex to implement
  - Probably easier to define ASI page-table so it doesn't fault during lockdown
- Instead just log a warning
  - Provide information about the fault so that ASI page-table can be augmented not to fault
  - Lockdown sequences are expected to be short so finding required mapping should be simple

KVM ASI

KVM ASI

- Goal: run VM and handle (most) VM exits without exiting ASI
- Create one KVM ASI per vcpu
  - KVM ASI from the same VM uses the same ASI lockdown tag
- KVM ASI page-table mappings
  - kvm and kvm-intel modules
  - various kernel structures (context_tracking, irq_stat, rcu_data, hrtimer_bases...)
  - kvm_vmx and vcpu_vmx structures
  - KVM IO buses and memslots

KVM ASI Usage

- KVM ASI is used when running a guest VCPU
  - KVM_RUN ioctl
  - Enter KVM ASI
  - Start KVM ASI lockdown
    - VMEnter
    - VMExit

    KVM ASI Lockdown
  - Stop KVM ASI lockdown
  - Run (most) VMExit handlers with KVM ASI
  - vcpu_run loop
- Early test (from RFCv2) showed that:
  - Most of loop iterations are done without exiting ASI
  - Most common VMExits are almost always processed without exiting ASI

ASI Test Driver

ASI Test Driver

- Kernel module and CLI for testing
- Create a Test ASI
- Test sequences:
  - Simple ASI enter/exit
  - Using printk() with ASI
  - Access a mapped or unmapped buffer in ASI
  - Receive an interrupt or NMI (or a NMI in an interrupt) in ASI
  - Scheduling a task using ASI in and out
  - ASI lockdown

ASI Introspection

- Driver/CLI can also be used to introspect and interact with an ASI
  - List ASI faults
  - Clear ASI faults
  - Toggle reporting of stack trace on ASI fault
  - List ASI mappings
  - Add ASI mapping
  - Clear ASI mapping
- By default actions are applied to the test ASI
- Actions can also be applied to a specified target ASI such as KVM ASI
  - Option available to list KVM ASIs

ASI printk() Test Example (1/2)

- Test printk() in ASI:

```
# asicmd printk
Test printk (sequence 1)
   - rv = 0 ; result = 0 ; asi inactive
   - expect = asi active
TEST FAILED - unexpected ASI state
```

- Test has failed because of an ASI fault:

```
# asicmd fault
  ADDRESS              COUNT  SYMBOL
  0xffffffff8110b5a1      1   vprintk_func+0x11/0xbc
```

- Debugger shows that the fault is due to printk_context not being mapped:

```
0xffffffff8110b5a1: mov %gs:0x7ef106f8(%rip),%eax  # 0x1bca0 <printk_context>
```


ASI printk() Test Example (2/2)

- Add missing mapping: (printk_context is a percpu buffer of size 4)

```
# asicmd mapadd percpu:0x1bca0:4
mapadd 1bca0/4/0 percpu
```

- Run the test again:

```
# asicmd printk
Test printk (sequence 1)
   - rv = 0 ; result = 0 ; asi inactive
   - expect = asi active
TEST FAILED - unexpected ASI state
```

- Test is still failing but because of a new fault: (cpu_number is not mapped)

```
# asicmd fault
  ADDRESS               COUNT   SYMBOL
  0xffffffff8110b5a1        1   vprintk_func+0x11/0xbc
  0xffffffff811081f3        1   log_store.constprop.27+0x1f3/0x280
```


KVM ASI Introspection Example (1/2)

- List target ASIs

```
# asicmd target
TARGET
kvm/1502/vcpu0
kvm/1502/vcpu1
```

- Dump KVM ASI Faults

```
# asicmd -t kvm/1502/vcpu0 fault
ADDRESS             COUNT  SYMBOL
0xffffffff8111b026  20878  rcu_qs+0x6/0x60
```

KVM ASI Introspection Example (2/2)

- Dump KVM ASI Mapping

```
# asicmd -t kvm/1502/vcpu0 map
ADDRESS                            SIZE  LEVEL
0xffff888033e88800                0x5c8    PTE
0xffff8880349c0a00                0x1e8    PTE
0xffff8880343ea000                0x468    PTE
0xffff88807d623ec8                 0x20    PTE
0xffff88807d623f18                 0x20    PTE
0xffff88807ce65a80                 0x80    PTE
0xffff88806e662a40                 0x50    PTE
0xffffc9000064c000               0x4000    PTE
0xffff88807d5c0000               0x2480    PTE
0xffff88806edc7000               0x1000    PTE
0xffff88806eddb000               0x1000    PTE
0xffff88806edda000               0x1000    PTE
0xffff88807c34b268               0x1000    PTE
0xffffea0001ed2dc0                 0x40    PMD
0xffff888079dc0000                0xa70    PTE
0xffff88807b4ad000               0x1000    PTE
...
```


ASI Status

ASI Status

- 4 RFCs already submitted
- RFCv5 in preparation, with 5 parts:
  - ASI Infrastructure and PTI
  - ASI Lockdown **[NEW]**
  - Decorated Page Table (DPT)
  - KVM ASI **[NEW]**
  - ASI Driver and CLI
- Core ASI code looks stable since RFCv3
- But need more testing of ASI Lockdown and KVM ASI
- Need to measure performance impact on KPTI and KVM

Thank You

Alexandre Chartre
alexandre.chartre@oracle.com

ORACLE