Hypervisor enforced security policies for NTOS, secure kernel and a child partition

tandasat.github.io/blog/2024/02/12/hyper-v-configs.html

Feb 12, 2024

This post aims to clarify security policies implemented by the Windows hypervisor for the root partition VTL 0 (NTOS), 1 (secure kernel), and a child partition (guest VM) by comparing their VMCSes on an Intel platform.

Summary

I start with the summary of my take, as the rest of this article is fairly “dry”.

The most interesting difference is VTL 1 having writable code. I heard of this but never verified it myself. I knew VTL 1 mapped UEFI runtime service code with the writable permission when the Memory Attributes Table was unavailable, but my target system did have it and properly implemented W^X (ref). I am unclear why code is left writable almost entirely. Similarly, it is questionable that IA32_EFER.NXE is not set for the VTL 1 guest.

The other intriguing part is largely accessible IO ports from VTL 0. I would have to study the functionality of these IO ports a bit more to be confident to say these are ok in this way. You may find the list of documented IO ports in volume 1 of the PCH specification, for example:

Memory Mapping—Intel® 600 Series Chipset Family On-Package PCH

Table 7. Fixed I/O Ranges Decoded by PCH

I/O Address	Read Target	Write Target	Internal Unit (unless[E]: External)[2]	Separate Enable/Disable
20h – 21h	Interrupt Controller	Interrupt Controller	Interrupt	None
24h – 25h	Interrupt Controller	Interrupt Controller	Interrupt	None

On MSRs, besides the undocumented MSRs, it is worth recreating the list on newer models as it might change depending on the existence of physical MSRs. Additionally, `IA32_SPEC_CTRL` being writable from the child partition is interesting. Could not a guest disable mitigation features and leak information? I would be curious to know.

On CR4, it is interesting that more bits are intercepted and shadowed for VTL 0 than the child partition. I cannot think of a reason off the top of my head.

It may be good security research to compare these with other hypervisor-protected systems. Is there a similar software architecture with a different setup, and would that imply overlooked security holes on that system or Windows? In addition to that, being intercepted by a hypervisor does not mean there is no chance of a bug; it is an attack surface to be inspected.

The rest of the post analyzes raw data.

Setup

I checked VMCS configurations on Windows 22H2 on the 9th generation Intel processor. The guest partition is Windows 22H2 with Hyper-V configuration version 11.0. HVCI is enabled for the root partition and disabled for the guest partition.

Comparison

MSRs

The lists of MSRs accessible without interception are the same between VTL 0 and 1. The child partition can access only a subset of these MSRs.

▼ Details

This is a list of writable MSRs for VTL 0 and 1. Ones writable from the child partition are marked with (G).

- 0x0 - IA32_P5_MC_ADDR
- 0x48 - IA32_SPEC_CTRL (G)
- 0x49 - IA32_PRED_CMD (G)
- 0xc5 - IA32_PMC4
- 0xc6 - IA32_PMC5
- 0xc7 - IA32_PMC6
- 0xc8 - IA32_PMC7
- 0xe2 - MSR_PKG_CST_CONFIG_CONTROL
- 0xe3 -
- 0xe7 - IA32_MPERF
- 0xe8 - IA32_APERF
- 0x10b - IA32_FLUSH_CMD (G)
- 0x17b - IA32_MCG_CTL
- 0x17f - MSR_ERROR_CONTROL
- 0x18a - IA32_PERFEVTSEL4
- 0x18b - IA32_PERFEVTSEL5

- 0x18c - IA32_PERFEVTSEL6
- 0x18d - IA32_PERFEVTSEL7
- 0x198 - IA32_PERF_STATUS
- 0x199 - IA32_PERF_CTL
- 0x19a - IA32_CLOCK_MODULATION
- 0x19b - IA32_THERM_INTERRUPT
- 0x19c - IA32_THERM_STATUS
- 0x19d - MSR_THERM2_CTL
- 0x1a2 - MSR_TEMPERATURE_TARGET
- 0x1ac - MSR_TURBO_POWER_CURRENT_LIMIT
- 0x1ad - MSR_TURBO_RATIO_LIMIT
- 0x1b0 - IA32_ENERGY_PERF_BIAS
- 0x1b1 - IA32_PACKAGE_THERM_STATUS
- 0x1b2 - IA32_PACKAGE_THERM_INTERRUPT
- 0x1fa - IA32_DCA_0_CAP
- 0x1fc - MSR_POWER_CTL
- 0x30c - IA32_FIXED_CTR3
- 0x30d - MSR_IQ_COUNTER1
- 0x30e - MSR_IQ_COUNTER2
- 0x30f - MSR_IQ_COUNTER3
- 0x310 - MSR_IQ_COUNTER4
- 0x311 - MSR_IQ_COUNTER5
- 0x312 -
- 0x313 -
- 0x314 -
- 0x315 -
- 0x316 -
- 0x317 -
- 0x318 -
- 0x329 - MSR_PERF_METRICS
- 0x4c5 - IA32_A_PMC4
- 0x4c6 - IA32_A_PMC5
- 0x4c7 - IA32_A_PMC6
- 0x4c8 - IA32_A_PMC7
- 0x601 - MSR_VR_CURRENT_CONFIG
- 0x609 -
- 0x60a - MSR_PKGC3_IRTL
- 0x60b - MSR_PKGC_IRTL1
- 0x60c - MSR_PKGC_IRTL2
- 0x610 - MSR_PKG_POWER_LIMIT
- 0x615 - PLATFORM_POWER_LIMIT

- 0x61e - MSR_PCIE_PLL_RATIO
- 0x620 - UNCORE_RATIO_LIMIT
- 0x621 - MSR_UNCORE_PERF_STATUS
- 0x64f - MSR_CORE_PERF_LIMIT_REASONS
- 0x65c - MSR_PLATFORM_POWER_LIMIT
- 0x6b0 - MSR_GRAPHICS_PERF_LIMIT_REASONS
- 0x6b1 - MSR_RING_PERF_LIMIT_REASONS
- 0x772 - IA32_HWP_REQUEST_PKG
- 0x773 - IA32_HWP_INTERRUPT
- 0x774 - IA32_HWP_REQUEST
- 0x777 - IA32_HWP_STATUS
- 0x17d1 - IA32_HW_FEEDBACK_CONFIG
- 0x17d2 - IA32_THREAD_FEEDBACK_CHAR
- 0x17da - IA32_HRESET_ENABLE
- 0xc0000100 - IA32_FS_BASE (G)
- 0xc0000101 - IA32_GS_BASE (G)
- 0xc0000102 - IA32_KERNEL_GS_BASE (G)

IO ports

The lists of IO ports accessible without interception are different between 3 configurations.

- For VTL 0, all ports except below are accessible:
  - 0x20, 0x21, 0xa0, 0xa1 - Master and Slave PIC (reference)
  - 0x64 - PS/2 Controller (reference)
  - 0xcf8, 0xcfc-0xcff - PCI config address and data (reference)
  - 0x1805 - (upper) PM1 control registers
- For VTL 1, all ports are accessible.
- For the child partition, none of the ports are accessible.

Memory

Below are a few observations with a quick look.

- For both VTL 0 and 1, translations are identity-mapped.
- For VTL 1, code is almost entirely writable even if HVCI is enabled for VTL 0.
- For the child partition, translations are simple offsets within a few large blocks of physical memory.

  For example, when GPA 0x0 is mapped to PA 0x224200000, GPA 0x4600000 is mapped to 0x228800000 (0x224200000 + 0x4600000).

Control fields

Pin-based VM-execution controls

There is no difference between the 3 configurations.

▼ Details
"1" means the feature is enabled.

VTL 0	VTL 1	Child	Bits
1	1	1	0 External-interrupt exiting
1	1	1	
1	1	1	
1	1	1	3 NMI exiting
1	1	1	
1	1	1	5 Virtual NMIs
0	0	0	6 Activate VMX preemption timer
0	0	0	7 Process posted interrupts

Primary processor-based VM-execution controls

There are a few differences.

- for VTL 1, "Interrupt-window exiting" is enabled
- for the child partition, MWAIT, MONITOR, and MOV-DR are intercepted
- for the child partition, all IO port access are intercepted

▼ Details
"1" means the feature is enabled.

VTL 0	VTL 1	Child	Bits
0	0	0	
1	1	1	
0	1	0	2 Interrupt-window exiting 🔔
1	1	1	3 Use TSC offsetting
1	1	1	
1	1	1	

VTL 0	VTL 1	Child	Bits
1	1	1	
1	1	1	7 HLT exiting
1	1	1	
0	0	0	9 INVLPG exiting
0	0	1	10 MWAIT exiting 🔔
1	1	1	11 RDPMC exiting
0	0	0	12 RDTSC exiting
1	1	1	
1	1	1	
0	0	0	15 CR3-load exiting
0	0	0	16 CR3-store exiting
0	0	0	17 Activate tertiary controls
0	0	0	
0	0	0	19 CR8-load exiting
0	0	0	20 CR8-store exiting
1	1	1	21 Use TPR shadow Setting
0	0	0	22 NMI-window exiting
0	0	1	23 MOV-DR exiting 🔔
0	0	1	24 Unconditional I/O exiting 🔔
1	1	0	25 Use I/O bitmaps 🔔
1	1	1	
0	0	0	27 Monitor trap flag
1	1	1	28 Use MSR bitmaps
0	0	1	29 MONITOR exiting 🔔
0	0	0	30 PAUSE exiting

VTL 0	VTL 1	Child	Bits
1	1	1	31 Activate secondary controls

Secondary processor-based VM-execution controls

There are a few differences:

- For the child partition, “WBINVD” is intercepted.
- “Mode-based execute control for EPT” is enabled only for VTL 0. This is because VTL 1 does not have as strict memory protection as VTL 0, and the child partition (VM) was not configured to enable HVCI.

▼ Details

“1” means the feature is enabled.

VTL 0	VTL 1	Child	Bits
1	1	1	0 Virtualize APIC accesses
1	1	1	1 Enable EPT
1	1	1	2 Descriptor-table exiting
1	1	1	3 Enable RDTSCP
0	0	0	4 Virtualize x2APIC mode
1	1	1	5 Enable VPID
0	0	1	6 WBINVD exiting 🔔
1	1	1	7 Unrestricted guest
0	0	0	8 APIC-register virtualization
0	0	0	9 Virtual-interrupt delivery
0	0	0	
0	0	0	11 RDRAND exiting
1	1	1	12 Enable INVPCID
0	0	0	13 Enable VM functions
0	0	0	14 VMCS shadowing
1	1	1	15 Enable ENCLS exiting

VTL 0	VTL 1	Child	Bits
0	0	0	16 RDSEED exiting
0	0	0	17 Enable PML
0	0	0	18 EPT-violation #VE
1	1	1	19 Conceal VMX from PT
1	1	1	20 Enable XSAVES/XRSTORS
0	0	0	21 PASID translation
1	0	0	22 Mode-based execute control for EPT 🔔
0	0	0	23 Sub-page write permissions for EPT
0	0	0	24 Intel PT uses guest physical addresses
0	0	0	25 Use TSC scaling
0	0	0	26 Enable user wait and pause
0	0	0	27 Enable PCONFIG
0	0	0	28 Enable ENCLV exiting
0	0	0	
0	0	0	30 VMM bus-lock detection
0	0	0	31 Instruction timeout

Primary VM-exit controls

For the child partition, “Load IA32_PAT” is enabled.

▼ Details

“1” means the feature is enabled.

VTL 0	VTL 1	Child	Bits
1	1	1	
1	1	1	
1	1	1	2 Save debug controls
1	1	1	

VTL 0	VTL 1	Child	Bits
1	1	1	
1	1	1	
1	1	1	
1	1	1	
1	1	1	
1	1	1	9 Host address-space size
1	1	1	
1	1	1	
0	0	0	12 Load IA32_PERF_GLOBAL_CTRL
1	1	1	
1	1	1	
1	1	1	15 Acknowledge interrupt on exit
1	1	1	
1	1	1	
0	0	0	18 Save IA32_PAT
0	0	1	19 Load IA32_PAT 🔔
0	0	0	20 Save IA32_EFER
0	0	0	21 Load IA32_EFER
0	0	0	22 Save VMX-preemption timer value
0	0	0	23 Clear IA32_BNDCFGS
1	1	1	24 Conceal VMX from PT
0	0	0	25 Clear IA32_RTIT_CTL
0	0	0	26 Clear IA32_LBR_CTL
0	0	0	27 Clear UINV
0	0	0	28 Load CET state

VTL 0	VTL 1	Child	Bits
0	0	0	29 Load PKRS
0	0	0	30 Save IA32_PERF_GLOBAL_CTL
0	0	0	31 Activate secondary controls

VM-entry controls

For the child partition, “Load IA32_PAT” is enabled.

▼ Details

“1” means the feature is enabled.

VTL 0	VTL 1	Child	Bits
1	1	1	
1	1	1	
1	1	1	2 Load debug controls
1	1	1	
1	1	1	
1	1	1	
1	1	1	
1	1	1	
1	1	1	
1	1	1	9 IA-32e mode guest
0	0	0	10 Entry to SMM
0	0	0	11 Deactivate dualmonitor treatment
1	1	1	
0	0	0	13 Load IA32_PERF_GLOBAL_CTRL
0	0	1	14 Load IA32_PAT 🔔
0	0	0	15 Load IA32_EFER
0	0	0	16 Load IA32_BNDCFGS

VTL 0	VTL 1	Child	Bits
1	1	1	17 Conceal VMX from PT
0	0	0	18 Load IA32_RTIT_CTL
0	0	0	19 Load UINV
0	0	0	20 Load CET state
0	0	0	21 Load guest IA32_LBR_CTL
0	0	0	22 Load PKRS

ENCLS-exiting bitmap

For the child partition, all `ENCLS` leaf functions are intercepted.

▼ Details

"1" means the leaf function is intercepted.

VTL 0	VTL 1	Child	Bits
0	0	1	ENCLS[ECREATE]
0	0	1	ENCLS[EADD]
1	1	1	ENCLS[EINIT]
0	0	1	ENCLS[EREMOVE]
0	0	1	ENCLS[EDBGRD]
0	0	1	ENCLS[EDBGWR]
0	0	1	ENCLS[EEXTEND]
0	0	1	ENCLS[ELDB]
0	0	1	ENCLS[ELDU]
0	0	1	ENCLS[EBLOCK]
0	0	1	ENCLS[EPA]
0	0	1	ENCLS[EWB]
0	0	1	ENCLS[ETRACK]
0	0	1	ENCLS[EAUG]

VTL 0	VTL 1	Child	Bits
0	0	1	ENCLS[EMODPR]
0	0	1	ENCLS[EMODT]
0	0	1	ENCLS[ERDINFO]
0	0	1	ENCLS[ETRACKC]
0	0	1	ENCLS[ELDBC]
0	0	1	ENCLS[ELDUC]

Exception bitmap

There is no difference between the 3 configurations.

▼ Details

"1" means the exception is intercepted.

VTL 0	VTL 1	Child	Bits
0	0	0	Divide Error Exception
1	1	1	Debug Exception
1	1	1	NMI Interrupt
0	0	0	Breakpoint Exception
0	0	0	Overflow Exception
0	0	0	BOUND Range Exceeded Exception
0	0	0	Invalid Opcode Exception
0	0	0	Device Not Available Exception
0	0	0	Double Fault Exception
0	0	0	Coprocessor Segment Overrun
0	0	0	Invalid TSS Exception
0	0	0	Segment Not Present
0	0	0	Stack Fault Exception
0	0	0	General Protection Exception

VTL 0	VTL 1	Child	Bits
0	0	0	Page-Fault Exception
0	0	0	
0	0	0	x87 FPU Floating-Point Error
0	0	0	Alignment Check Exception
1	1	1	Machine-Check Exception
0	0	0	SIMD Floating-Point Exception
0	0	0	Virtualization Exception
0	0	0	Control Protection Exception

CR0 guest/host mask

There is no difference between the 3 configurations.

▼ Details

“1” means access to the bit position is intercepted and shadowed.

VTL 0	VTL 1	Child	Bits
1	1	1	0 Protection Enable
0	0	0	1 Monitor Coprocessor
0	0	0	2 Emulation
0	0	0	3 Task Switched
0	0	0	4 Extension Type
1	1	1	5 Numeric Error
1	1	1	
1	1	1	
1	1	1	
1	1	1	
1	1	1	
1	1	1	

VTL 0	VTL 1	Child	Bits
1	1	1	
1	1	1	
1	1	1	
1	1	1	
1	1	1	16 Write Protect
1	1	1	
1	1	1	18 Alignment Mask
1	1	1	
1	1	1	
1	1	1	
1	1	1	
1	1	1	
1	1	1	
1	1	1	
1	1	1	
1	1	1	
1	1	1	
1	1	1	29 Not Write-through
1	1	1	30 Cache Disable
1	1	1	31 Paging

CR4 guest/host mask

For VTL 0, several bits are intercepted and shadowed.

▼ Details

“1” means access to the bit position is intercepted and shadowed.

VTL 0	VTL 1	Child	Bits

VTL 0	VTL 1	Child	Bits
1	0	0	0 Virtual-8086 Mode Extensions 🔔
1	0	0	1 Protected-Mode Virtual Interrupts 🔔
1	0	0	2 Time Stamp Disable 🔔
1	0	0	3 Debugging Extensions 🔔
1	1	1	4 Page Size Extensions
1	1	1	5 Physical Address Extension
1	1	1	6 Machine-Check Enable
0	0	0	7 Page Global Enable
0	0	0	8 Performance-Monitoring Counter Enable
1	0	0	9 Operating System Support for FXSAVE and FXRSTOR instructions 🔔
1	0	0	10 Operating System Support for Unmasked SIMD Floating-Point Exceptions 🔔
1	1	1	11 User-Mode Instruction Prevention
1	1	1	12 57-bit linear addresses
1	1	1	13 VMX-Enable Bit
1	1	1	14 SMX-Enable Bit
1	1	1	
1	1	1	16 FSGSBASE-Enable Bit
1	1	1	17 PCID-Enable Bit
1	1	1	18 XSAVE and Processor Extended States-Enable Bit
1	1	1	19 Key-Locker-Enable Bit
1	1	1	20 SMEP-Enable Bit
1	1	1	21 SMAP-Enable Bit
1	1	1	22 Enable protection keys for user-mode pages
1	1	1	23 Control-flow Enforcement Technology
1	1	1	24 Enable protection keys for supervisor-mode pages

VTL 0	VTL 1	Child	Bits
1	1	1	25 User Interrupts Enable Bit

Call for actions

Besides the open questions I made above, there are opportunities to find new vulnerabilities in the Windows hypervisor if you extend hvext.js for AMD platforms. I discovered two vulnerabilities specific to the Intel platforms while writing the tool, so I would not be surprised if similar issues exist on AMD platforms.

Reference: steps to get them

1. Enable hypervisor debugging and get hvext.js working.

2. Reduce the number of logical processors to 1 and reboot. This makes VTL 0, 1 and guest transitions tremendously clearer.

   ```
   > bcdedit /set numproc 1
   ```

3. To break on VMCS switching, we need to set breakpoints on the all `VMPTRLD` instructions in the hypervisor image. For this, get the range of hypervisor's .text section first.

   ```
   kd> lm
   start             end                 module name
   fffff863`87673000 fffff863`87a75000   hv         (no symbols)

   kd> !dh -s fffff863`87673000
   ...
   SECTION HEADER #9
     .text name
     19C0C4 virtual size
     200000 virtual address
     19D000 size of raw data
   ...
   ```

4. Then, search the `VMPTRLD` instructions in the range with the `#` command.

   ```
   kd> # vmptrld fffff863`87673000+200000 L 19C0C4
   ...
   ```

5. Finally, set a breakpoint for each discovered instruction.

   Note that there were 41 instances of the `VMPTRLD` instructions in the version I tested, and Windbg could set only up to 30 breakpoints. However, this was not a big issue as only 4 of them were used during the regular operation. To figure out which instructions are used, you can trace execution of them instead of breaking in each time with commands like this:

   ▼ Details

```
; Offsets are valid only for the version 10.0.22621.2861

bp hv+0x20af68 ".echo ' 0'; dp rcx+188h l1; gc"
bp hv+0x2123cf ".echo ' 1'; dp rcx+188h l1; gc"
bp hv+0x216c2e ".echo ' 2'; dp rcx+188h l1; gc"
bp hv+0x21a174 ".echo ' 3'; dp rcx+188h l1; gc"
bp hv+0x22093b ".echo ' 4'; dp rcx+188h l1; gc"      ; used to switch VTL 0 and 1
bp hv+0x22b377 ".echo ' 5'; dp rcx+188h l1; gc"
bp hv+0x22c1ba ".echo ' 6'; dp rcx+188h l1; gc"
bp hv+0x22c6f4 ".echo ' 7'; dp rcx+188h l1; gc"
bp hv+0x22cd17 ".echo ' 8'; dp rcx+188h l1; gc"      ; used to switch guest and VTL 0
bp hv+0x239401 ".echo ' 9'; dp rsp+30h l1; gc"
bp hv+0x248112 ".echo '10'; dp rcx+188h l1; gc"
bp hv+0x25589a ".echo '11'; dp rcx+188h l1; gc"
bp hv+0x2559ae ".echo '12'; dp rcx+188h l1; gc"
bp hv+0x33e1d3 ".echo '13'; dp rcx+188h l1; gc"
bp hv+0x33e2f5 ".echo '14'; dp rcx+188h l1; gc"
bp hv+0x33ead1 ".echo '15'; dp rcx+188h l1; gc"
bp hv+0x340e8d ".echo '16'; dp r8+118h l1; gc"
bp hv+0x340eed ".echo '17'; dp rsp+58h l1; gc"
bp hv+0x3410e2 ".echo '18'; dp rcx+118h l1; gc"
bp hv+0x341a6e ".echo '19'; dp rbp+48h l1; gc"
bp hv+0x347146 ".echo '20'; dp rcx+29A20h l1; gc"    ; used only for the first launch
bp hv+0x34960c ".echo '21'; dp rcx+188h l1; gc"
bp hv+0x34971f ".echo '22'; dp rcx+188h l1; gc"
bp hv+0x34985d ".echo '23'; dp rcx+188h l1; gc"
bp hv+0x349acd ".echo '24'; dp r8+188h l1; gc"
bp hv+0x349c95 ".echo '25'; dp rcx+188h l1; gc"
bp hv+0x34b8b8 ".echo '26'; dp r8+118h l1; gc"
bp hv+0x34b8f9 ".echo '27'; dp rsp+58h l1; gc"
bp hv+0x34ba7f ".echo '28'; dp rcx+188h l1; gc"
bp hv+0x34baed ".echo '29'; dp rsp+50h l1; gc"
bp hv+0x34cf28 ".echo '30'; dp rcx+188h l1; gc"
bp hv+0x34f3f4 ".echo '31'; dp rax+188h l1; gc"
bp hv+0x34f4e4 ".echo '32'; dp rax+188h l1; gc"
bp hv+0x34feae ".echo '33'; dp rcx+188h l1; gc"
bp hv+0x352070 ".echo '34'; dp rcx+188h l1; gc"
bp hv+0x352100 ".echo '35'; dp rcx+188h l1; gc"
bp hv+0x3521d9 ".echo '36'; dp rcx+188h l1; gc"
bp hv+0x352b9d ".echo '37'; dp rcx+188h l1; gc"
bp hv+0x352bb0 ".echo '38'; dp rdx+0B0h l1; gc"
bp hv+0x3541a5 ".echo '39'; dp rcx+188h l1; gc"      ; used only during start up
bp hv+0x391b62 ".echo '40'; dp rcx+188h l1; gc"
```