## Contents

1 Introduction

2 AMD Processor Bugs

- Bug ID: 731 – Incorrect IOMMU Page Table Translation
- Bug ID: 385 – Incorrect Address for an L3 Cache Error Machine Check
- Bug ID: 77 – CALLF OR JMPF Fails to Signal General Protection Fault
- Bug ID: 573 – FSINCOS Causes Incorrect Instruction Pointer Update
- Bug ID: 564 – Failing Auto-Halt Restart in SMM Save State
- Bug ID: 561 – Incorrect Page Table Walk
- Bug ID: 578 – Incorrect Processor Behavior through Branch Prediction
- Bug ID: 639 – Incorrect Instruction Pointer Following CALL
- Bug ID: 784 – Incorrect Value from Control Register Data Resulting from Load Instruction
- Bug ID: 503 – Incorrect APIC Task-Priority Register
- Bug ID: 691 – 1MB L3 Subcache Processors Execute WBINVD incorrectly
- Bug ID: 704 – Interrupt or Debug Trap Causes Storage of Incorrect Instruction Pointer
- Bug ID: 744 – Trap Registers not Restored from CC6 power state transition
- Bug ID: 776 – Consecutive Linear Page Shift of Instruction Pointer Causes Incorrect Branch Prediction
- Bug ID: 144 – CFLUSH Fails to Invalidate Cached Shadow RAM Addresses
- Bug ID: 770 – I/O C-state Request Clears Guest Current Privilege Level
- Bug ID: 401 – Incorrect RIP Saved After Software Interrupt
- Bug ID: 238 – Page Invalidation Failure During Page Splintering
- Bug ID: 418 – Incorrect SVM Nested Page Translation in PAE Mode
- Bug ID: 440 – Corrupted CR3 in System Management Mode
- Bug ID: 342 – System Management Interrupts May Cause Guest VMs to Run With Interrupts Disabled
- Bug ID: 171 – Processor Incorrectly Executes under Mixed Guest/Host VM States Following an Instruction Breakpoint on VMRUN
- Bug ID: 563 – Resume Flag Incorrectly Set for Instruction Breakpoints on the CLI Instruction
- Bug ID: 738 – VMCB Control Block Stack Pointer Corrupted After Concurrent Debug Exception and System Management Interrupts
- Bug ID: 659 – Interrupt Shadow Field Not Correctly Cleared under VMCB
- Bug ID: 734 – Data Corruption under VM Guest Execution
- Bug ID: 165 – Some VM State Information Lost Following Guest Mode VMEXIT
- Bug ID: 170 – Execution of Incorrect Byte Code on Exit From Guest Mode

Acknowledgements
Abstract

This report is a qualitative study of security-critical processor bugs observed in AMD processors. Processor technology updates expand hardware feature sets which, similar to software, may introduce new design or implementation flaws, or bugs, into the hardware. Processor manufacturers release documentation for these bugs; in this study, we classify security-critical processor bugs by the methods we could employ with the Secure Virtual Architecture (SVA) to mitigate these bugs.
Chapter 1

Introduction

This document describes AMD processor bugs marked as security-critical by Hicks et al. [5], meaning that each bug may have a negative impact on system security if triggered. These bugs are documented in the AMD Errata Documents [7, 11, 12, 14, 13, 10, 9, 8]. We categorize each bug by the deterrence technique(s) that we believe would effectively prevent software from exploiting the bug. We propose to mitigate these bugs using Secure Virtual Architecture (SVA) [4].

As Figure 1.1 shows, SVA [4] is a compiler-based virtual machine interposed between the software stack and the processor. All software is compiled to a virtual instruction set and is translated by SVA to the processor’s native instruction set. During translation, SVA can statically analyze code and/or instrument code with run-time checks to enforce security policies. Additionally, SVA provides a set of instructions dubbed SVA-OS which provide a hardware abstraction layer that operating system kernels can use to configure privileged processor state and manipulate program state, e.g. for context switching.

![Figure 1.1: Secure Virtual Architecture](image)

This report classifies security-critical processor bugs in the AMD errata based on how SVA could mitigate the bug. Our deterrence categories are as follows:

**SVA-OS:** Some processor bugs can be mitigated by enhancing the implementation of the SVA-OS instructions as the bugs are triggered by configuring privileged hardware state in some specific manner. SVA-OS can control I/O device configuration, Memory Management Unit (MMU) configuration, and interrupt handler configuration [3]. The SVA virtual instruction set does not provide a means of configuring x86 Model-Specific Registers (MSRs) [1]; however, the SVA-OS instruction set can be easily extended to provide controlled interfaces for configuring the processor features that are, in the native instruction set, controlled by MSRs.

**Run-time Checks:** Another set of deterrence techniques is the introduction of run-time checks. A run-time check is defined as any additional code that SVA inserts into the native code during native code generation. Run-time checks verify that the next operation to be performed does not trigger a processor bug. Inserting run-time checks is a method of last resort because it introduces run-time overhead.

**Grammar Checks:** Some processor bugs are triggered through unsafe sequences of instructions which may simply be avoided through transformations during code translation. Grammar checks statically analyze the native code that SVA generates to guarantee the safety of execution. Our static analysis procedure
generates a context free grammar and an equivalent Pushdown Automaton (PDA) which represent all possible
dynamic instruction sequences possible during program execution. Intersecting this PDA with a Finite State
Automaton (FSA) constructed from a regular language describing unsafe instruction sequences that can
trigger a particular processor bug produces a final PDA describing all unsafe execution paths within the
analyzed program.

**SVA Cannot Mitigate:** Bugs in this category cannot be deterred by SVA. At the level which SVA can
operate and perform its analysis, an insufficient amount of processor state information is available to SVA
during abnormal execution caused by processor bugs.

**Insufficient Documentation:** Official documentation on known processor bugs is provided by the pro-
cessor manufacturer. For some bugs, the manufacturer has provided insufficient details on the bug for us to
determine how SVA could mitigate the bug.

**No Hardware Virtualization Support:** Several processor bugs involve new hardware features such as
hardware virtualization [6]. SVA does not yet provide instructions for software to configure and use hardware
virtualization features. Any bugs involving this hardware cannot be triggered in current SVA-based systems
but will need to be mitigated in future SVA-based systems that provide support for hardware virtualization
features.

The remainder of this document will describe security-critical AMD processor bugs, deterrence catego-
rizations, and mitigations. Each entry has (1) a unique Bug ID provided by the manufacturer, (2) our short
title, (3) a manufacturer-provided description of how the particular processor bug is triggered and its effects,
(4) the deterrence category selected (from those described above), (5) a planned policy to mitigate the bug,
and (6) a reference to the original documentation of the bug.
Chapter 2

AMD Processor Bugs

Table 2.1 shows how many bugs we placed into each mitigation category:

<table>
<thead>
<tr>
<th>Mitigation Type</th>
<th>Number of Bugs</th>
<th>Percentage of Bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVA-OS</td>
<td>7</td>
<td>25%</td>
</tr>
<tr>
<td>Run-time Checks</td>
<td>3</td>
<td>11%</td>
</tr>
<tr>
<td>Grammar Checks</td>
<td>2</td>
<td>7%</td>
</tr>
<tr>
<td>SVA Cannot Mitigate</td>
<td>1</td>
<td>4%</td>
</tr>
<tr>
<td>Insufficient Documentation</td>
<td>2</td>
<td>7%</td>
</tr>
<tr>
<td>No Hardware Virtualization Support</td>
<td>13</td>
<td>46%</td>
</tr>
</tbody>
</table>

Table 2.1: Hardware Bug Categorization Statistics

A description of each bug, with one bug per page, now follows.
Bug ID: 731 – Incorrect IOMMU Page Table Translation

**Description:** “The processor may perform incorrect IOMMU translations using an IOMMU v1 page table that has a page size of 512 GB.”

**Deterrence Category:** SVA-OS Configuration

**Mitigation:** The SVA-OS I/O load/store instructions will need to be enhanced with checks to determine whether the I/O MMU is being misconfigured. Alternatively, we may need to add IOMMU instructions to SVA-OS to make the run-time checks efficient.

**Reference:** [http://developer.amd.com/wordpress/media/2012/10/48931.15h.Mod.10h-1Fh.Rev.Guide.pdf#page=57](http://developer.amd.com/wordpress/media/2012/10/48931.15h.Mod.10h-1Fh.Rev.Guide.pdf#page=57)
Bug ID: 385 – Incorrect Address for an L3 Cache Error Machine Check

Description: “The processor may report an incorrect address at NB Machine Check Address Register MSR0000_0412 when executing a machine check for an L3 cache error. In addition, when disabling an L3 cache index by writing to L3 Cache Index Disable Registers F3x[1C0, 1BC] [Index], the processor may not disable the intended L3 cache index.” [9]

Deterrence Category: SVA-OS Configuration

Mitigation: The SVA Virtual Machine will set register F3x1B8[23] to 1b upon boot. The SVA-OS virtual instruction set will not provide an interface for software to change this configuration setting during regular operation.

Bug ID: 77 – CALLF OR JMPF Fails to Signal General Protection Fault

**Description:** “If the target selector of a far call or far jump (CALLF or JMPF) instruction references a 16-byte long mode system descriptor where any of the last 8 bytes are beyond the GDT or LDT limit, the processor fails to report a General Protection fault.”

**Deterrence Category:** SVA-OS Configuration

**Mitigation:** Upon boot, the SVA VM should ensure that it configures the GDT and LDT limits so that no descriptors “fall off” the end of the GDT/LDT. The SVA VM configures the GDT and LDT internally and does not provide instructions allowing software to change the GDT and LDT configurations.

**Reference:** http://support.amd.com/TechDocs/41322_10h_Rev_Gd.pdf#page=47
Bug ID: 573 – FSINCOS Causes Incorrect Instruction Pointer Update

**Description:** “After execution of an FSINCOS instruction, the processor core may incorrectly update the instruction pointer (rIP) and execute incorrect instructions or may hang.”

**Deterrence Category:** Grammar Check

**Mitigation:** The SVA VM’s Code Generator should not generate any FSINCOS instructions; instead, it should generate sequential FSIN and FCOS instructions. The grammar checks static analysis can verify that the code generator has done this correctly.

**Reference:** http://support.amd.com/TechDocs/41788_11h_Rev_Gd.pdf#page=48
Bug ID: 564 – Failing Auto-Halt Restart in SMM Save State

**Description:** “The processor core may not set the auto-halt restart flag (offset FEC9h of the SMM save state area) when the HLT instruction causes the processor core to enter the core C6 (CC6) state and is then interrupted by an SMI. After the SMM code executes the RSM instruction, the processor core does not re-enter the HLT or CC6 state due to this incorrect auto-halt restart flag.” [10]

**Deterrence Category:** Run-time Check

**Mitigation:** The SVA virtual to native code translator should insert code during code generation that manually sets the auto-halt restart flag (offset FEC9h of the SMM save state area [10]) before a HLT instruction and resets it to its original unmodified value immediately following the HLT instruction.

**Reference:** http://support.amd.com/TechDocs/44739_12h_Rev_Gd.pdf#page=40
Bug ID: 561 – Incorrect Page Table Walk

**Description:** “When HWCR[TlbCacheDis] (MSRC001_0015[3]) is 1b and the page table resides in I/O or DRAM that is marked non-cacheable, the processor may incorrectly perform page table walks.”

**Deterrence Category:** SVA-OS Configuration

**Mitigation:** Set HWCR[TlbCacheDis] to 0b. The SVA virtual instruction set will not provide an interface for modifying HWCR[TlbCacheDis] during regular operation, forcing it to remain set to 0b.

**Reference:** http://support.amd.com/TechDocs/47534_14h_Mod_00h-0Fh_Rev_GUIDe.pdf#page=44
Bug ID: 578 – Incorrect Processor Behavior through Branch Prediction

**Description:** “Under a highly specific and detailed set of internal timing conditions involving multiple events occurring within a small window of time, the processor branch prediction logic may cause the processor core to decode incorrect instruction bytes.” [11]

**Deterrence Category:** Insufficient Documentation

**Mitigation:** Not Applicable

**Reference:** http://support.amd.com/TechDocs/47534_14h_Mod_00h-0Fh_Rev_Guide.pdf#page=49
Bug ID: 639 – Incorrect Instruction Pointer Following CALL

Description: “Under a highly specific and detailed set of internal timing conditions where the processor has performed at least 100 pushes, pops, near-calls and/or near-returns without executing any other operation that uses the stack pointer, the processor may incorrectly execute the following instruction:

- CALL RSP without and offset (instruction encoding FFD4h).

Under the above conditions, the processor does not execute the CALL instruction and instead may treat this instruction as if it is a NOP (no operation) instruction. The processor incorrectly updates the rIP to the address following the CALL. This CALL RSP instruction should transfer instruction execution to the stack (i.e. it changes the rIP to the value that was in the rSP prior to the execution of the instruction). If the stack address is marked with a no-execute attribute (the NX bit is set in the page table), this generates a #GP exception. The stack is commonly marked with a no-execute attribute. As a result, the use of this instruction encoding is uncommon in applications.” [11]

Deterrence Category: Grammar Check

Mitigation: Design a Regular Language to characterize at least 100 pushes/pops/near-calls/near-returns (excluding any other stack pointer operations), followed by a CALL RSP.

Reference: http://support.amd.com/TechDocs/47534_14h_Mod_00h-0Fh_Rev_Guide.pdf#page=59
Bug ID: 784 – Incorrect Value from Control Register Data Resulting from Load Instruction

Description: “Under a highly specific and detailed set of internal timing conditions, the processor core may provide control register data as the result of an instruction that is performing a load from memory. In order to observe this incorrect data, the processor must be at the highest privilege level (CPL 0) and be speculatively or non-speculatively executing certain invalid opcodes.” [1]

Deterrence Category: Run-time Check

Mitigation: As this bug can only be triggered when executing invalid opcodes, the control-flow integrity (CFI) instrumentation in SVA [2] can ensure that no invalid opcodes are ever executed.

Reference: http://support.amd.com/TechDocs/47534_14h_Mod_00h-0Fh_Rev_Guide.pdf#page=71
Bug ID: 503 – Incorrect APIC Task-Priority Register

Description: “An APIC task priority register (TPR) write may use an incorrect internal buffer for the data.”

Deterrence Category: SVA-OS Configuration


Reference: http://support.amd.com/TechDocs/48063_15h_Mod_00h-0Fh_Rev_Guide.pdf#page=25
Bug ID: 691 – 1MB L3 Subcache Processors Execute WBINVD incorrectly

**Description:** “The processor may fail to flush the full address range of L3 cache when executing a **WBINVD** instruction, or **INVD** instruction with Hardware Configuration Register[INVDWBINVD] = 1b (MSRC001_0015[4]). This occurs when the L3 cache is less than 8MB per northbridge and is configured using at least one 1MB L3 subcache, as indicated by the L3SubcacheSize fields (L3 Cache Parameter Register[L3SubcacheSize[3:0]], D18F3x1C4[[15:12],[11:8],[7:4],[3:0]] = Dh or Eh).” [14]

**Deterrence Category:** SVA Cannot Mitigate

**Mitigation:** Not Applicable

**Reference:** http://support.amd.com/TechDocs/48063_15h_Mod_00h-0Fh_Rev_Guide.pdf#page=65
Bug ID: 704 – Interrupt or Debug Trap Causes Storage of Incorrect Instruction Pointer

**Description:** “Under a highly specific and detailed set of internal timing conditions, the processor may store an incorrect instruction pointer (rIP) while processing an interrupt or debug trap exception (#DB).”

**Deterrence Category:** Insufficient Documentation

**Mitigation:** Not Applicable

**Reference:** [http://support.amd.com/TechDocs/48063_15h_Mod_00h-0Fh_Rev_Guide.pdf#page=70](http://support.amd.com/TechDocs/48063_15h_Mod_00h-0Fh_Rev_Guide.pdf#page=70)
Bug ID: 744 – Trap Registers not Restored from CC6 power state transition

Description: “Following a core C6 (CC6) power state transition, the processor core may not restore the following registers:

- MSRC001_0053, IO Trap Register 3
- MSRC001_0052, IO Trap Register 2
- MSRC001_0051, IO Trap Register 1
- MSRC001_0050, IO Trap Register 0

Instead, the registers are set to zero when the processor resumes from CC6 state.

This erratum only applies on processor models that have a single core per compute-unit (Compute Unit Status Register[DualCore], D18F5x80[16] is 0b).”

Deterrence Category: SVA-OS Configuration

Mitigation: Force trap registers MSRC001_005[0:3] to be disabled by setting MSRC001_0054 bits 7, 5, 3 and 1 to 0b. The SVA virtual instruction set will not provide instructions for modifying MSRC001_0054.

Bug ID: 776 – Consecutive Linear Page Shift of Instruction Pointer Causes Incorrect Branch Prediction

**Description:** “Under a highly specific and detailed set of internal timing conditions, the processor core may incorrectly fetch instructions when the instruction pointer (rIP) changes (via a branch or other call, return) between two consecutive linear address 4K pages with the same offset in rIP[11:6].” 

**Deterrence Category:** SVA-OS Configuration

**Mitigation:** Configure SVA to set MSRC001_1021[26] to 1b. The SVA virtual instruction set will not provide an interface for software to modify MSRC001_1021[26] during regular operation.

**Reference:** [http://support.amd.com/TechDocs/51810_16h_00h-0Fh_Rev_Guide.pdf#page=24](http://support.amd.com/TechDocs/51810_16h_00h-0Fh_Rev_Guide.pdf#page=24)
Bug ID: 144 – CFLUSH Fails to Invalidate Cached Shadow RAM Addresses

Description: “WrDram and RdDram bits in extended MTRR type registers are used to copy BIOS ROM to corresponding DRAM and then execute out of DRAM. When these are configured to direct writes to ROM (WrMem = 0b) and reads to DRAM (RdMem = 01b), the CFLUSH instruction will not invalidate shadow RAM addresses in the cache.” [8]

Deterrence Category: Run-time Check

Mitigation: When translating the cache flushing virtual instructions to native code, the SVA VM code generator should generate code that checks to see if the address to flush is in shadow RAM and, if it is, uses the WBINVD instruction instead of the CFLUSH instruction to flush the cache line. According to the bug report [8], the WBINVD instruction will correctly flush cache lines for shadow RAM.

Reference: https://support.amd.com/TechDocs/33610.PDF#page=42
Bug ID: 770 – I/O C-state Request Clears Guest Current Privilege Level

**Description:** “Following an unintercepted guest access to an I/O address that causes an entry to a C-state, the processor may enter core C6 (CC6) state and incorrectly clear the guest current privilege level (CPL) to zero.” [11]

**Deterrence Category:** No Hardware Virtualization Support

**Mitigation:** Not Applicable

**Reference:** http://support.amd.com/TechDocs/47534_14h_Mod_00h-0Fh_Rev_Guide.pdf#page=69
Bug ID: 401 – Incorrect RIP Saved After Software Interrupt

**Description:** “The processor does not use VMCB.NextRIP when event injection is used on an INTn, INT3, or INTO software interrupt. In the event that the injected instruction encounters a fault, the processor may incorrectly store the address following the software interrupt on the stack.”

**Deterrence Category:**  No Hardware Virtualization Support

**Mitigation:** Not Applicable

**Reference:** http://support.amd.com/TechDocs/41788_11h_Rev_Gd.pdf#page=46
Bug ID: 248 – Page Invalidation Failure During Page Splintering

Description:  “When an address mapped by a guest uses a larger page size than the host, the TLB entry created uses the size of the smaller page; this is referred to as page splintering. TLB entries that are the result of page splintering may not be invalidated when the large page is invalidated in the guest using INVLPGA.” [11]

Deterrence Category:  No Hardware Virtualization Support

Mitigation:  Not Applicable

Reference:  http://support.amd.com/TechDocs/47534_14h_Mod_00h-0Fh_Rev_Guide.pdf#page=69
Bug ID: 418 – Incorrect SVM Nested Page Translation in PAE Mode

Description:  “The processor may use an incorrect cached copy of translation tables during an SVM nested page translation when the host is in legacy Physical Address Extension (PAE) mode and the guest address translation tables reside in physical page zero.” [11]

Deterrence Category:  No Hardware Virtualization Support

Mitigation:  Not Applicable

Reference:  http://support.amd.com/TechDocs/41322_10h_Rev_Gd.pdf#page=128
Bug ID: 440 – Corrupted CR3 in System Management Mode

Description: “The processor writes bits 47:32 as 0000h of SMM Save State offset FF38h (Host CR3) when all of the following conditions are met:

- An SMI occurs while in a guest context.
- SMIs are not intercepted to the hypervisor and can use a direct transition from the guest to SMM mode.
- Nested paging is in use (VMCB offset 090h[0], NP_ENABLE, is 1b). The SVM Host CR3 address is greater than 4GB (VMCB Offset 0B0h, N_CR3, bits 47:32 are non-zero).
- Guest is not in long mode at the time of the SMI (guest Extended Feature Enable Register EFER[Long Mode Enable], MSRC000_0080[8], is 0b).

After the SMM BIOS executes an RSM instruction, the processor may then use this incorrect host CR3 for guest operation.” [11]

Deterrence Category: No Hardware Virtualization Support

Mitigation: Not Applicable

Reference: http://support.amd.com/TechDocs/41322_10h_Rev_Gd.pdf#page=135
Bug ID: 342 – System Management Interrupts May Cause Guest VMs to Run With Interrupts Disabled

Description: “During a resume from SMM that is due to an unintercepted SMI from a SVM guest context, the processor core does not restore the correct effective interrupt flag (IF) if the guest VMCB V_INTR_MASKING bit (offset 060h bit 24) is 1b. Under these conditions, the effective interrupt flag may be zero. SMIs are not intercepted if VMCB offset 00Ch bit 2 is 0b or HWCR[SmmLock] (MSRC001_0015[0]) is 1b.” [1]

Deterrence Category: No Hardware Virtualization Support

Mitigation: Not Applicable

Reference: http://support.amd.com/TechDocs/41322_10h_Rev_Gd.pdf#page=83
Bug ID: 171 – Processor Incorrectly Executes under Mixed Guest/Host VM States Following an Instruction Breakpoint on VMRUN

**Description:** “VMRUN can be interrupted using a hardware instruction breakpoint using one of the debug registers, DR[0-3]. When the debug handler executes IRET, the processor is expected to execute the VMRUN instruction. However, in the failing case, the processor incorrectly re-enters the breakpoint handler with mixed guest and host state. This in turn causes erroneous execution and leads to unpredictable system behavior.” [1]

**Deterrence Category:** No Hardware Virtualization Support

**Mitigation:** Not Applicable

Bug ID: 563 – Resume Flag Incorrectly Set for Instruction Breakpoints on the CLI Instruction

Description: “A processor core may incorrectly store rFLAGS.RF = 1 in the VMCB when the processor is performing an SVM interception on a CLI instruction, including an intercept on an exception that occurred during execution of the CLI instruction.” [11]

Deterrence Category: No Hardware Virtualization Support

Mitigation: Not Applicable

Reference: http://support.amd.com/TechDocs/41322_10h_Rev_Gd.pdf#page=83
Bug ID: 738 – VMCB Control Block Stack Pointer Corrupted After Concurrent Debug Exception and System Management Interrupts

Description: “During processing of a RSM instruction to return from a system management interrupt (SMI) that occurred while in secure virtual machine (SVM) mode and without the SMI being intercepted, the processor core may not restore the guest stack pointer prior to presenting a debug exception (#DB) on the resumed guest instruction. In the event that this #DB is intercepted (or some other exception during the #DB processing is intercepted), the stack pointer from system management mode (SMM) may be saved into the virtual machine control block (VMCB offset 1D8h). There is no error if an interception does not occur during the #DB.” [11]

Deterrence Category: No Hardware Virtualization Support

Mitigation: Not Applicable

Reference: http://support.amd.com/TechDocs/47534_14h_Mod_00h-0Fh_Rev_Guide.pdf#page=67
Bug ID: 659 – Interrupt Shadow Field Not Correctly Cleared under VMCB

Description: "The processor may fail to clear the VMCB INTERRUPT_SHADOW field (VMCB offset 068h bit 0) when intercepting or interrupting an SVM guest that is executing a Move String instruction with a REP prefix under interrupt shadow. This erratum does not occur on the last iteration of the Move String instruction." [11]

Deterrence Category: No Hardware Virtualization Support

Mitigation: Not Applicable

Reference: http://support.amd.com/TechDocs/48063_15h_Mod_00h-0Fh_Rev_Guide.pdf#page=51
Bug ID: 734 – Data Corruption under VM Guest Execution

Description: “Under a highly specific and detailed set of internal timing conditions during a #VMEXIT for a virtual machine guest that has multiple virtual CPUs, the processor may store incorrect data to the virtual machine control block (VMCB) reserved and guest save areas and may also store outside of the VMCB.”

Deterrence Category: No Hardware Virtualization Support

Mitigation: Not Applicable

Reference: http://support.amd.com/TechDocs/48063_15h_Mod_00h-0Fh_Rev_Guide.pdf#page=83
Bug ID: 165 – Some VM State Information Lost Following Guest Mode VMEXIT

Description: “If VMRUN returns with #VMEXIT(INVALID), the EVENTINJ field in the VMCB is unconditionally cleared.”

Deterrence Category: No Hardware Virtualization Support

Mitigation: Not Applicable

Reference: https://support.amd.com/TechDocs/33610.PDF#page=57
Bug ID: 170 – Execution of Incorrect Byte Code on Exit From Guest Mode

Description: “On an exit from guest mode (world switch) when CR3 changes, under a highly specific and detailed set of conditions, incorrect code bytes may be forwarded from the prefetch buffer.” [1]

Deterrence Category: No Hardware Virtualization Support

Mitigation: Not Applicable

Reference: https://support.amd.com/TechDocs/33610.PDF#page=62
Acknowledgements

This work was funded in part by National Science Foundation Award Number 1463870. The authors thank Joel Seiferas from the University of Rochester for discussions that led toward the development of our study’s grammar checking static analysis approach.
Bibliography


[13] Inc., A. M. D. Revision guide for AMD family 16h models 00h-0fh processors, June 2013.

[14] Inc., A. M. D. Revision guide for AMD family 15h models 00h-0fh processors, September 2014.