White Paper

PROTECTING VM REGISTER STATE WITH SEV-ES

February 17, 2017
David Kaplan

DISCLAIMER

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale.

Introduction

Recently, AMD introduced the **Secure Encrypted Virtualization (SEV)** technology [1] that integrates memory encryption with AMD-V virtualization to provide support for encrypted virtual machines (VMs). Encrypted virtual machines are ideal for multi-tenant environments such as cloud computing as they enable protection from a variety of cross-VM and hypervisor-based attacks. For instance, a hypervisor bug which enables a co-resident VM to escape its sandbox and read arbitrary memory on the system cannot be used to steal data or compromise an SEV-enabled VM.

While no security system is 100% secure, SEV significantly reduces the attack surface of VMs in the cloud by encrypting a VM's memory with a key unique to that VM and unknown to the hypervisor. Many secrets and important information are typically stored in a VM's memory space and encrypting this content helps prevent attacks and leakage of sensitive data.

However, when secrets are being actively used by the VM they are often resident in CPU registers as well as memory. Whenever a VM stops running, due to an interrupt or other event, its register contents are saved to hypervisor memory and this memory is readable by the hypervisor even if SEV is enabled. This information could allow a malicious or compromised hypervisor to steal information or alter critical values in guest state such as an instruction pointer, encryption key, etc.

The new **SEV with Encrypted State (SEV-ES)** feature blocks attacks like these by encrypting and protecting all CPU register contents when a VM stops running. This prevents the leakage of information in CPU registers to components like the hypervisor, and can even detect and prevent malicious modifications to CPU register state. SEV-ES builds upon SEV to provide an even smaller attack surface and additional protection for a guest VM from the hypervisor.

This document presents a technical overview of the SEV-ES feature, the principles behind the architecture, and protections offered to further isolate encrypted VMs. For additional technical details, please see the AMD64 Programmer's Manual [2].

Security Goals

The SEV-ES technology is intended to protect a guest VM from attacks on its register state from a malicious hypervisor. These attacks may include reading guest register values, writing malicious values, or even replaying old state back into the VM. Reading guest register values from a malicious hypervisor can result in silent data exfiltration, for instance by reading the XMM registers typically used to hold AES keys when using the x86 AES instructions. Writing malicious values could be used to directly or indirectly modify the control flow of the guest VM, for instance by overwriting the RIP when resuming a VM. Such a modification could lead to potentially unintended behavior inside the guest VM, such as skipping a critical security check. SEV-ES encrypts and integrity protects the guest VM's register state to protect against these types of attacks.

The challenge with protecting VM register state lies in the fact that sometimes hypervisors do need access to VM registers for purposes of providing services such as device emulation, MSR handling, etc. These accesses must be carefully controlled to prevent malicious use, which can be difficult to detect.

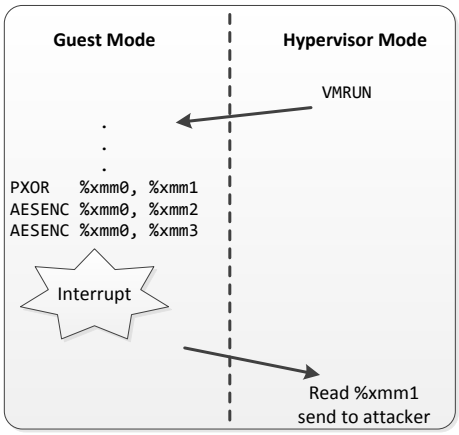

Figure 1: Stealing AES keys from a VM

To address this, the SEV-ES technology enables the guest VM to decide what information it chooses to expose to the hypervisor on a per-case basis. This is similar in principle to SEV where guest VMs are in control of which pages of memory they choose to share. By placing this functionality inside the guest VM, it is both flexible and customizable for different scenarios.

Technical Overview

Encrypted Register State

In the legacy AMD-V architecture, saving and restoring guest VM register state is a multi-step process. The VMRUN instruction used to transfer control to a guest VM only saves and loads a subset of overall machine state [2]. Many pieces of state, including system registers such as TR as well as general purpose registers like RDX, are not automatically saved and restored by the VMRUN instruction. Before resuming a guest, a typical hypervisor will also load the guest's GPR values, use the VMLOAD instruction to load additional system state, and possibly even load the guest floating point register state with XRSTOR.

The SEV-ES architecture combines this entire operation into a single atomic hardware instruction, VMRUN. When VMRUN is executed for a guest with SEV-ES enabled, the CPU hardware loads all guest register information including system registers, GPRs, and floating point registers. Similarly when the VM stops running (a so-called "VMEXIT"), all of this state is saved automatically by hardware back to memory and hypervisor state is loaded. This single atomic process ensures that the world switch between a hypervisor and guest cannot be interrupted and information from the guest cannot leak into the hypervisor.

When hardware saves and restores the guest register state it does so to memory encrypted with the VM's memory encryption key. This key is not known to any software on the CPUs and cannot be used by the hypervisor so the hypervisor is not able to read the actual guest register state. Furthermore, when saving register state the CPU computes an integrity-check value. This integrity-check value is saved in protected DRAM that is not accessible to any CPU software. It is checked when the guest is later resumed with VMRUN by the CPU to ensure that guest register state has not been tampered with by the hypervisor.

The SEV-ES architecture therefore ensures that not only can the guest register contents be kept secret from the hypervisor, but also that the hypervisor cannot easily manipulate or replay them. The integrity-check value will detect such a modification to the register state since the last VMEXIT and the CPU will refuse to resume a guest VM if the integrity-check value is not correct. In other words, the only state to which a guest VM can be resumed is to the exact same state it was last in.

Note that in the SEV-ES architecture, the Virtual Machine Control Block (VMCB) used to describe a particular VM is divided into two sections. The first section is called the "control area" and is owned and managed by the hypervisor. The control section includes information about what events the hypervisor wishes to intercept, interrupt delivery information, etc. The second section, or "save area" is used to store the VM register state. When SEV-ES is enabled, this section is encrypted as described above and integrity protected.

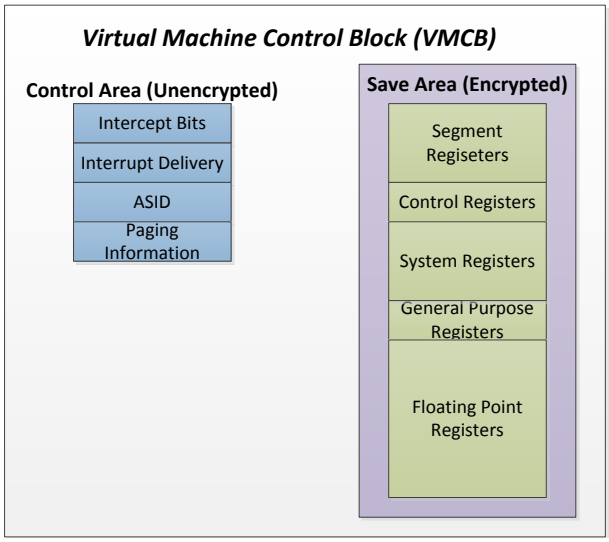

Figure 2: Virtual Machine Control Block

VM "Exits"

After a VMRUN instruction is executed, the CPU hardware continues executing the guest VM until an exit event occurs, referred to as a VMEXIT. The specific events that may result in a VMEXIT are defined by the AMD64 architecture [2] and are configured by the hypervisor setting "intercept bits" in the control section of the VMCB. For example, the hypervisor may set bits to configure a VMEXIT to occur on external interrupts, writes to control registers, reads from specific ports, etc. In the traditional AMD-V architecture, when a VMEXIT event occurs the CPU hardware returns control back to the hypervisor with an event code indicating what caused the VMEXIT.

In the SEV-ES architecture, the set of possible VMEXIT events are divided into two groups referred to as Automatic Exits (AE) and Non-Automatic Exits (NAE). In general, NAE events occur when the guest VM does something that will require hypervisor emulation (e.g. MMIO, MSR access, etc.). In contrast, AE events do not require any hypervisor emulation and include asynchronous interrupts, shutdown events, and certain types of page faults.

When SEV-ES is enabled, AE events are the only VMEXIT events which cause a full world switch and transfer control back to the hypervisor. These events cause the CPU hardware to save and encrypt all guest register state and load the hypervisor state as described in the earlier section. After performing whatever tasks the hypervisor desires, it can resume the guest with a VMRUN instruction which will transfer control back to the guest at the point where it was suspended.

Unlike AE events, NAE events always occur due to specific behavior within the guest such as executing particular instructions, accessing emulated device registers, etc. Unlike traditional AMD-V virtualization, these events do not cause a world switch back to the hypervisor when SEV-ES is enabled. Instead, when an NAE event occurs a new exception #VC (VMM Communication Exception) is generated and must be handled by the guest VM.

VMM Communication Exception (#VC)

The new #VC exception informs the guest VM operating system that it performed an event which requires hypervisor emulation. The #VC exception handler must then decide how to respond and request

appropriate services from the hypervisor. To facilitate this communication, the SEV-ES architecture defines a Guest Hypervisor Communication Block (GHCB). The GHCB resides in page of shared memory so it is accessible to both the guest VM and the hypervisor. The structure of the GHCB is not explicitly defined in the SEV-ES architecture, but is recommended to mirror the VMCB save area to enable the guest and hypervisor to easily communicate state information.

Different events require different pieces of state to be communicated between the guest and the hypervisor. For this reason, the #VC handler must determine at runtime which pieces of guest state it should expose to the hypervisor. For instance, if the #VC handler is invoked because the guest tried to execute a CPUID instruction, the guest should share the RAX value used with the hypervisor in order to receive CPUID emulation. Alternatively, if the guest attempted to write to a port with the OUTW instruction, it should share both its AX and DX values, etc. To share the required information, the #VC handler copies the relevant pieces of state and request for hypervisor services to the GHCB. In this way, the #VC handler is responsible for choosing what state it will expose to the hypervisor.

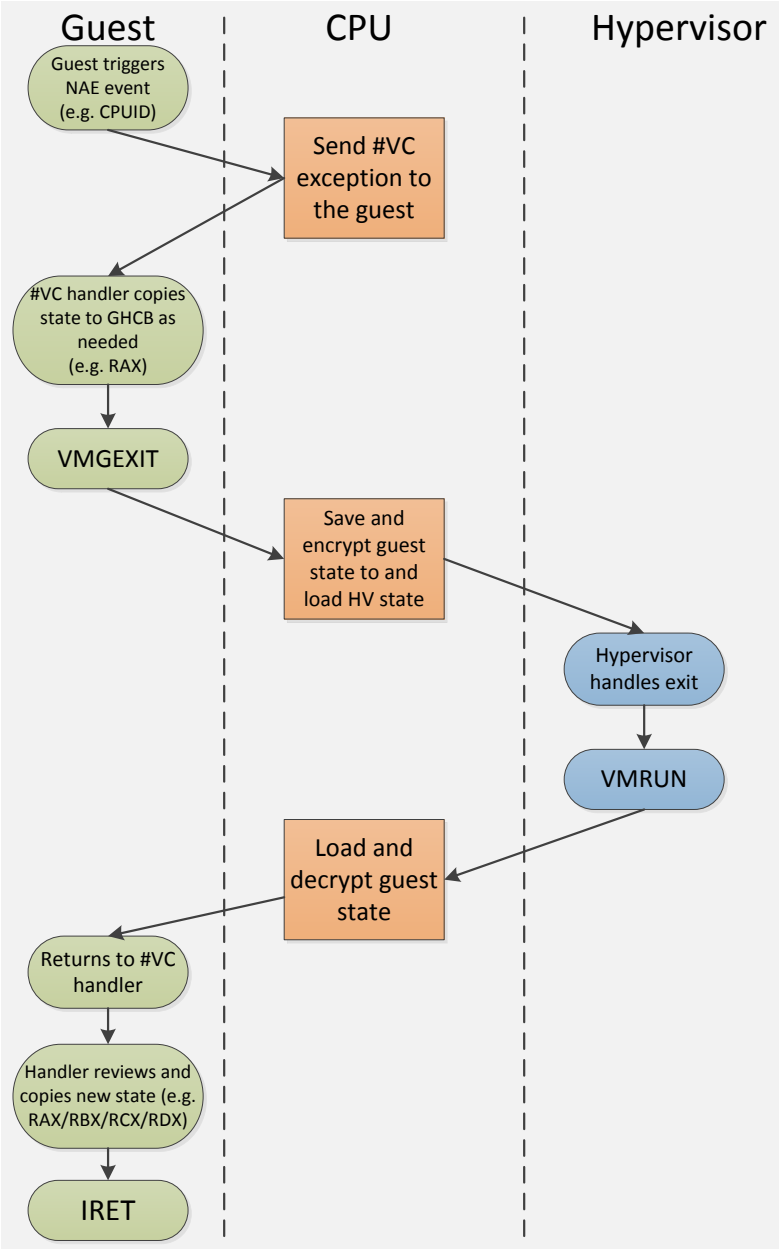

Figure 3: NAE Example Flow

After copying relevant state, the #VC handler transfers control to the hypervisor with the new VMGEXIT (Virtual Machine General Exit) instruction. This instruction results in an AE exit which saves all guest state, and resumes execution of the hypervisor.

At this point the hypervisor reads the GHCB to determine the emulation support required by the guest. As the hypervisor cannot directly modify guest state, it should perform the emulation required and place any new state values for the guest back into the GHCB. For instance, after a CPUID emulation the hypervisor should place the new values of RAX/RBX/RCD/RDX into the GHCB.

When the hypervisor resumes the guest, execution continues where it left off after the VMGEXIT instruction, still inside the #VC handler. The handler may now inspect the state changes requested by the hypervisor in the GHCB and determine if they are acceptable. If so, the handler copies the new state into the relevant registers and finishes the exception handler. An example of this #VC handler flow is shown in Figure 3: NAE Example Flow.

Initializing SEV-ES VMs

The initialization of SEV-ES VMs is very similar to existing SEV VMs [3]. Note that with SEV-ES, both the initial memory image as well as the initial CPU register state must be encrypted by the AMD Secure Processor (AMD-SP) before execution of the guest VM can start. During this initialization process, the memory image and initial CPU register state is measured cryptographically by the AMD-SP to generate a launch receipt that may be used for attestation of the guest. This attestation enables the owner of the guest VM to determine if the VM started successfully with the correct image and register state prior to releasing it secrets.

Software Impact

As with the SEV feature, no application code changes are required to support SEV-ES and only the guest operating system and hypervisor are impacted. In particular, the guest VM operating system must support handling the new #VC exception and communication with the hypervisor to achieve the emulation support required. Note that because all NAE events result in the new #VC exception, drivers within the guest VM operating system are not required to be modified or be SEV-ES aware. All communication with the hypervisor for emulation purposes is accomplished through a single centralized handler.

Because of the lack of visibility into guest register state with SEV-ES, hypervisor software must support the new GHCB structure for communication with the guest #VC handler. In many cases, this involves simply reading/writing to the GHCB structure instead of the VMCB in response to emulation requests. Furthermore, in SEV-ES many emulation related tasks (such as instruction cracking/re-execution) are actually moved into the #VC handler inside the guest meaning the hypervisor emulation support required is reduced substantially.

Performance Optimizations

While the primary goal of SEV-ES is to protect guest register state and put the guest in charge of what can be accessed, the #VC exception presents an opportunity to optimize guest/hypervisor communication. In particular, the guest #VC handler can potentially implement logic to reduce world switches and/or handle certain NAE events completely within the guest. For instance, the #VC handler can cache static values (e.g. CPUID results), or batch multiple hypervisor requests into a single VMGEXIT. Support for such optimizations may require special communication between the guest and hypervisor.

Conclusion

The SEV-ES feature provides additional hardware enforced security for isolating guest VMs from the hypervisor. It builds upon the memory protection offered by SEV to deliver confidentiality and integrity protection for guest register state to protect against information leakage and control flow manipulation by the hypervisor. SEV-ES empowers guest VMs to control which pieces of state the hypervisor can view and modify at all times, enabling support for existing functionality such as device emulation without requiring major changes to device drivers and hypervisors.

While SEV reduced the attack surface of VMs through memory encryption, protecting guest register state takes this protection a step further to offer more comprehensive protection from a compromised hypervisor.

References

[1] AMD Memory Encryption Whitepaper: http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf

[2] AMD64 Programmer's Manual Volume 2: http://support.amd.com/TechDocs/24593.pdf

[3] Secure Encrypted Virtualization Key Management: http://support.amd.com/TechDocs/55766_SEV-KM%20API_Specification.pdf