WHY NOT TRUST THE HYPERVERSOR?

**Guest Perspective**
- Hypervisor is code I don’t control
- I can’t tell if the hypervisor is compromised
- I have obligations to protect my data
  - Financial data
  - Customer data
  - Health records
  - Etc.
- I want assurances of privacy

**Hypervisor Perspective**
- I need to convince more customers to use my services
- I don’t want to be able to see what customers are doing
- I want to provide assurances of privacy
- I want to limit my customer’s exposures to bugs

Security is an important concern for cloud growth
## Hardware Memory Encryption - Attacks

**Physical Access Attacks**
- Probe the physical DRAM interface
- Install HW device that accesses guest memory
- Freeze then steal DIMMs
- Steal NVDIMMs

**Admin Access Attacks**
- Administrator scrapes memory of guest data areas
- Administrator injects code into a guest VM
- Hypervisor bug allows hosted guest to steal data from other guests

**Defended by AMD SME + SEV**
AMD HARDWARE MEMORY ENCRYPTION
SECURE ENCRYPTED VIRTUALIZATION (SEV) - COMPONENTS

**AMD Secure Processor**
- Manages keys during VM lifecycle
- Runs SEV API firmware
  - [https://developer.amd.com/wp-content/resources/55766.PDF](https://developer.amd.com/wp-content/resources/55766.PDF)

**Hypervisor**
- Calls SEV API as needed for VM management
- Is protected with its own key

**Guest OS**
- Chooses which pages to encrypt via page tables

**Example Development Environment**
- KVM/QEMU Hypervisor
- Linux Kernel for Guest
- OVMF (UEFI) for Guest BIOS
- SEV API (run in the AMD Secure Processor Firmware)
<video>
## SEV DEVELOPMENT UPDATE

<table>
<thead>
<tr>
<th>Feature</th>
<th>Supported Version</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SME (Host Encryption)</td>
<td>Linux 4.14</td>
<td></td>
</tr>
<tr>
<td>SEV (Guest Support)</td>
<td>Linux 4.15</td>
<td>OVMF &gt;= July 6, 2018</td>
</tr>
<tr>
<td>SEV (KVM HV Support)</td>
<td>Linux 4.16/QEMU 2.12</td>
<td>Libvirt 4.5</td>
</tr>
<tr>
<td>VirtIO support</td>
<td>Linux 4.15</td>
<td>Not including virtio-gpu</td>
</tr>
<tr>
<td>VirtIO-GPU support</td>
<td>Linux 4.19/QEMU 3.1</td>
<td></td>
</tr>
<tr>
<td>SEV Guest Migration</td>
<td>-</td>
<td>Available on AMD Github</td>
</tr>
<tr>
<td>SEV Tool</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>SEV Save &amp; Restore</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>SEV-ES Support</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

- Ubuntu 18.04, Fedora 28, SLES 15 all are supported as SEV guests
- Fedora 28 and SLES 15 are supported as SEV hypervisors
World switches now swap ALL register state
- Includes all segment registers, GPRs, FPU state (see Table B-4 in APM Vol2)
- All register state is encrypted with the guest encryption key
- Integrity value is calculated and stored in a protected page

The guest is notified by a new exception (#VC) when certain events occur
- The guest decides what state (if any) to share with the HV
- The guest invokes the HV to perform the required tasks
- The guest updates its state based on the output from the HV

The guest and HV use a special structure to communicate
- Guest-Hypervisor Communication Block (GHCB)
- Location set by guest, mapped as unencrypted memory page
## TYPES OF EXITS

### Automatic Exits (AE)
- Events that occur asynchronously to the guest (e.g. interrupts)
- Events that do not require exposing guest state (e.g. HLT)
- Nested page faults not due to MMIO emulation
- AE events save all state and exit to HV
- Only action HV can do is just resume the guest w/o modifications

### Non-Automatic Exits (NAE)
- All other exit events
- NAE events cause a #VC instead of a VMEXIT
- Guest handler may invoke the HV via VMGEXIT instruction

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Notes</th>
<th>HW Advances</th>
</tr>
</thead>
<tbody>
<tr>
<td>52h</td>
<td>VMEXIT_MC</td>
<td>Machine check exception</td>
<td>No</td>
</tr>
<tr>
<td>60h</td>
<td>VMEXITINTR</td>
<td>Physical INTR</td>
<td>No</td>
</tr>
<tr>
<td>61h</td>
<td>VMEXIT_NMI</td>
<td>Physical NMI</td>
<td>No</td>
</tr>
<tr>
<td>63h</td>
<td>VMEXIT_INIT</td>
<td>Physical INIT</td>
<td>No</td>
</tr>
<tr>
<td>64h</td>
<td>VMEXIT_VINTR</td>
<td>Virtual INTR</td>
<td>No</td>
</tr>
<tr>
<td>77h</td>
<td>VMEXIT_PAUSE</td>
<td>PAUSE instruction</td>
<td>Yes</td>
</tr>
<tr>
<td>78h</td>
<td>VMEXIT_HLT</td>
<td>HLT instruction</td>
<td>Yes</td>
</tr>
<tr>
<td>7Fh</td>
<td>VMEXIT_SHUTDOWN</td>
<td>Shutdown</td>
<td>No</td>
</tr>
<tr>
<td>8Fh</td>
<td>VMEXIT_EFER_WRITE_TRAP</td>
<td>Write to EFER</td>
<td>Yes</td>
</tr>
<tr>
<td>90h-</td>
<td>VMEXIT_CR[0-15]_WRITE_TRAP</td>
<td>Write to CRx</td>
<td>Yes</td>
</tr>
<tr>
<td>400h</td>
<td>VMEXIT_NPF</td>
<td>Only if PFCODE[3]=0 (no reserved bit error)</td>
<td>No</td>
</tr>
<tr>
<td>403h</td>
<td>VMEXIT_VMGEXIT</td>
<td>VMGEXIT instruction</td>
<td>Yes</td>
</tr>
<tr>
<td>-1</td>
<td>VMEXIT_INVALID</td>
<td>Invalid guest state</td>
<td>-</td>
</tr>
</tbody>
</table>
NAE FLOW EXAMPLE

**CPUID**

- Guest triggers VMEXIT condition
- Read ErrorCode => CPUID
- Write CPUID_EXIT to GHCB
- Copy RAX to GHCB

**CPU HW**

- Save/encrypt guest state
- Load HV state
- HV handles exit
- VMRUN

**Hypervisor**

- Save HV state
- Load/decrypt guest state
- #VC handler modifies state
- IRET
- Read GHCB => see CPUID_EXIT
- Read RAX, emulate CPUID
- Write RAX/RBX/RCX/RDX to GHCB

Copy RAX/RBX/RCX/RDX to register state
To attempt to standardize guest<->hypervisor interfaces, AMD has distributed a proposed GHCB Software Specification (see [https://developer.amd.com/sev/](https://developer.amd.com/sev/)).

**GHCB Specification defines:**
- Layout of GHCB memory page (4kb)
- What #VC exceptions guests are expected to handle (super-set of all supported HV intercepts)
- What values guests are expected to provide to the HV for each #VC
- What the HV is expected to provide on each VMGEXIT
- How SEV-ES guests are to be bootstrapped (including multi-vCPU environments)
- NMI handling for SEV-ES guests

**Goal:** To the greatest extent possible, provide a unified interface across all guest OS’s and hypervisors that support SEV-ES
## GHCB Protocol

<table>
<thead>
<tr>
<th>NAE Event</th>
<th>State to Hypervisor</th>
<th>State from Hypervisor</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSR_PROT (RDMSR)</td>
<td>RCX</td>
<td>RAX</td>
<td>XCR0 is only required to be supplied when a request for CPUID 0000_000D is made.</td>
</tr>
<tr>
<td></td>
<td>SW_EXITCODE=0x7c</td>
<td>RDX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SW_EXITINFO1=0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SW_EXITINFO2=0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPUID</td>
<td>RAX</td>
<td>RAX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RCX</td>
<td>RBX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>XCR0 (for RAX=0xd)</td>
<td>RCX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SW_EXITCODE=0x72</td>
<td>RDX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SW_EXITINFO1=0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SW_EXITINFO2=0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See proposed GHCB document for more details.
FEEDBACK SO FAR

- GHCB Specification is still undergoing review, comments are welcome!

- Comments:
  - GHCB memory layout matching VMCB layout -- Keeps hypervisor logic uniform, just changes where it gets values
  - Required registers for most intercepts -- Hypervisors are very uniform in this area
  - GHCB protocol negotiation -- Allow guest code to determine SEV-ES support at runtime

- Specific cases:
  - VMMCALL
  - AP startup
  - Debug support
  - SMM support
CONTAINERS WITH SEV
HYPERVERSOR-BASED RUNTIME WITH SEV

Docker with runC (default)

Docker with Kata Containers & SEV
DEMO: SEV RUNTIME
DROP-IN REPLACEMENT FOR RUNC

Docker with runc (default)

Docker with SEV containers
CONCLUSION

- Upstream SEV work is progressing across multiple projects
- SEV-ES hardware is available today, software still in progress
- GHCB specification attempting to unify guest<->hypervisor para-virt interfaces
- Exploring other uses of SEV, such as with Kata Containers

References:
- AMD SEV info (https://developer.amd.com/sev/)
- Github (with getting started scripts): https://github.com/AMDESE/AMDSEV
- Kata Container prototype: https://github.com/AMDESE/AMDSEV/tree/kata
- AMD64 Architecture Programmer’s Manual Volume 2: System Programming (sections 7.10 and 15.34)
- Secure Encrypted Virtualization Key Management
The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION

© 2018 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. ARM and Cortex are registered trademarks of ARM Limited in the UK and other countries. Other names are for informational purposes only and may be trademarks of their respective owners.