UNMAPPED PRIVATE MEMORY

Michael Roth

LINUX PLUMBERS – 2022

[Public]

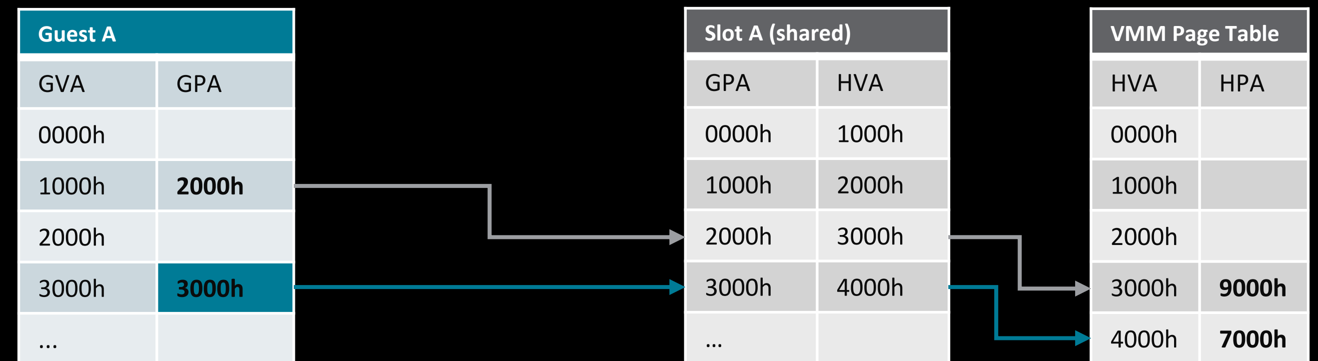
UNMAPPED PRIVATE MEMORY

- Proposed kernel infrastructure to back confidential guests with pages that are not mappable/accessible by userspace
- Generally synonymous with Chao Peng’s private memslot patchset:
  - “KVM: mm: fd-based approach for supporting KVM guest private memory”
- Proposed by a number of a developers for various reasons, but the most prevalent driver is TDX support, where writes to private guest memory by userspace result in #MC
- Also being evaluated for use with SEV-SNP, pKVM, and possibly others
- Description/topics here are somewhat SEV-SNP centric, but can hopefully still be extrapolated to some of these other use cases

AMD

UPM - PRIVATE MEMSLOTS

- Currently both shared/private memory are backed by normal memslots
  - private memory can be mapped into userspace just like normal memory
  - malloc() / mmap() →
- Adds new private memslot struct
  - Provides both shared/private memory
  - private memory allocated separately via memfd
  - memfd uses MFD_INACCESSIBLE
    - Not readable/writable
    - Can't be mmap()'d into userspace
- KVM MMU uses an xarray to determine whether to map guest memory from shared/private pool

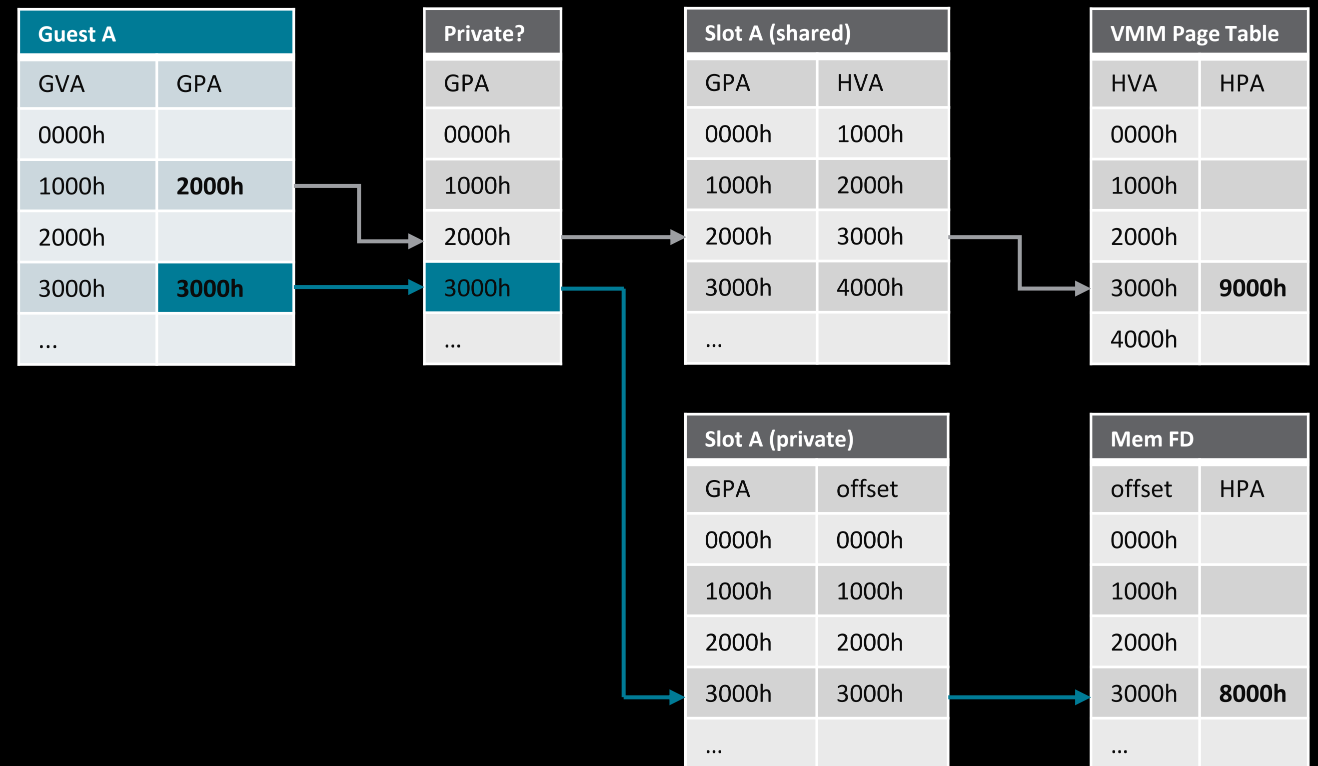


AMD

[Public]

UPM – PRIVATE MEMSLOTS

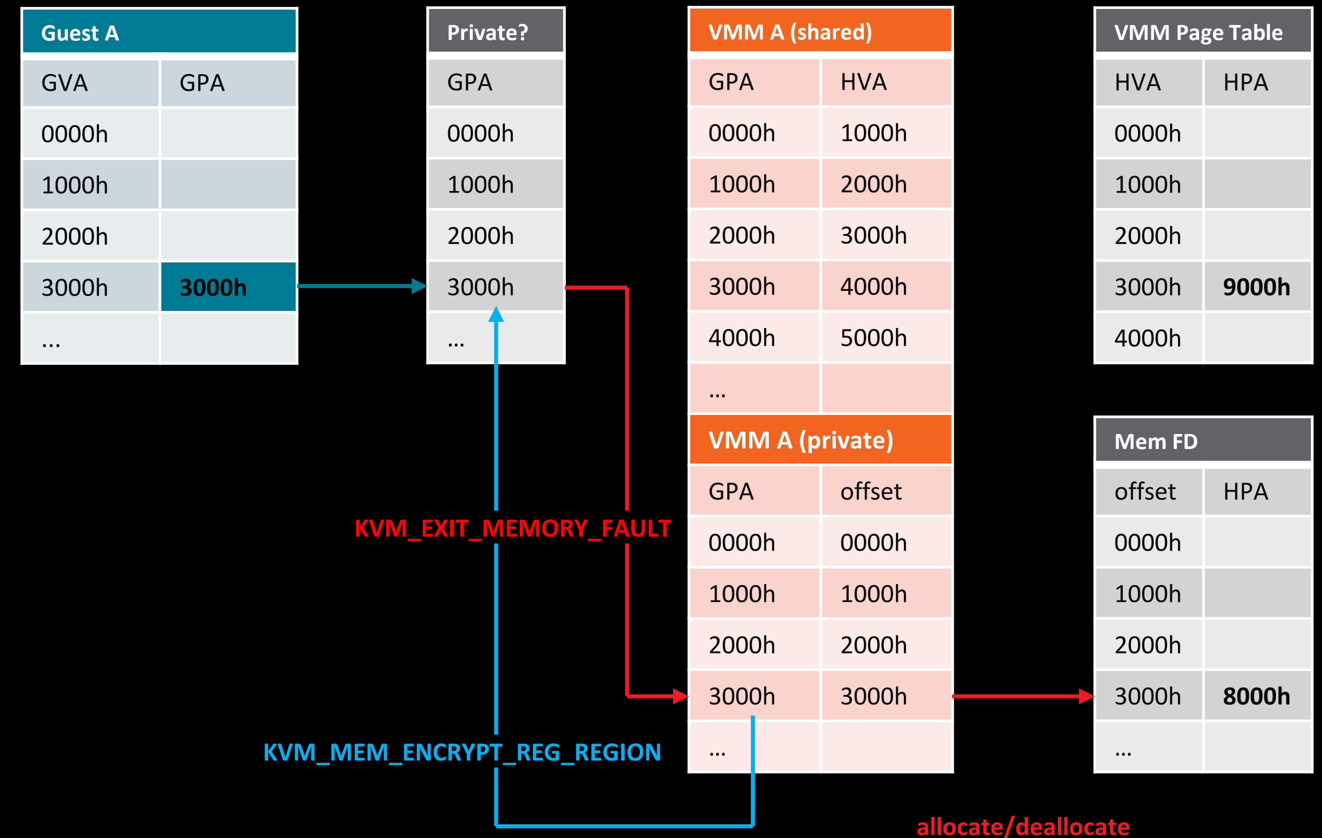
- Currently both shared/private memory are backed by normal memslots
  - private memory can be mapped into userspace just like normal memory
  - malloc() / mmap()
- Adds new private memslot struct
  - Provides both shared/private memory
  - private memory allocated separately via memfd
  - memfd uses MFD_INACCESSIBLE
    - Not readable/writable
    - Can't be mmap()'d into userspace
- KVM MMU uses an xarray to determine whether to map guest memory from shared/private pool



AMD

UPM – IMPLICIT CONVERSIONS

- KVM MMU uses an xarray to determine whether to map guest memory from shared/private pool
  - xarray controlled purely by userspace
    - KVM_MEM_ENCRYPT_REG_REGION
    - KVM_MEM_ENCRYPT_UNREG_REGION
- Implicit conversion
  - if C-bit does not match xarray state:
    - KVM_EXIT_MEMORY_FAULT
    - alloc/dealloc private/shared memory
    - VMM converts using REG/UNREG ioctl
- Explicit conversion
  - GHCB page-state change request forwarded to userspace
    - KVM_EXIT_VMGEXIT
    - alloc/dealloc private/shared memory
    - VMM converts using REG/UNREG ioctl

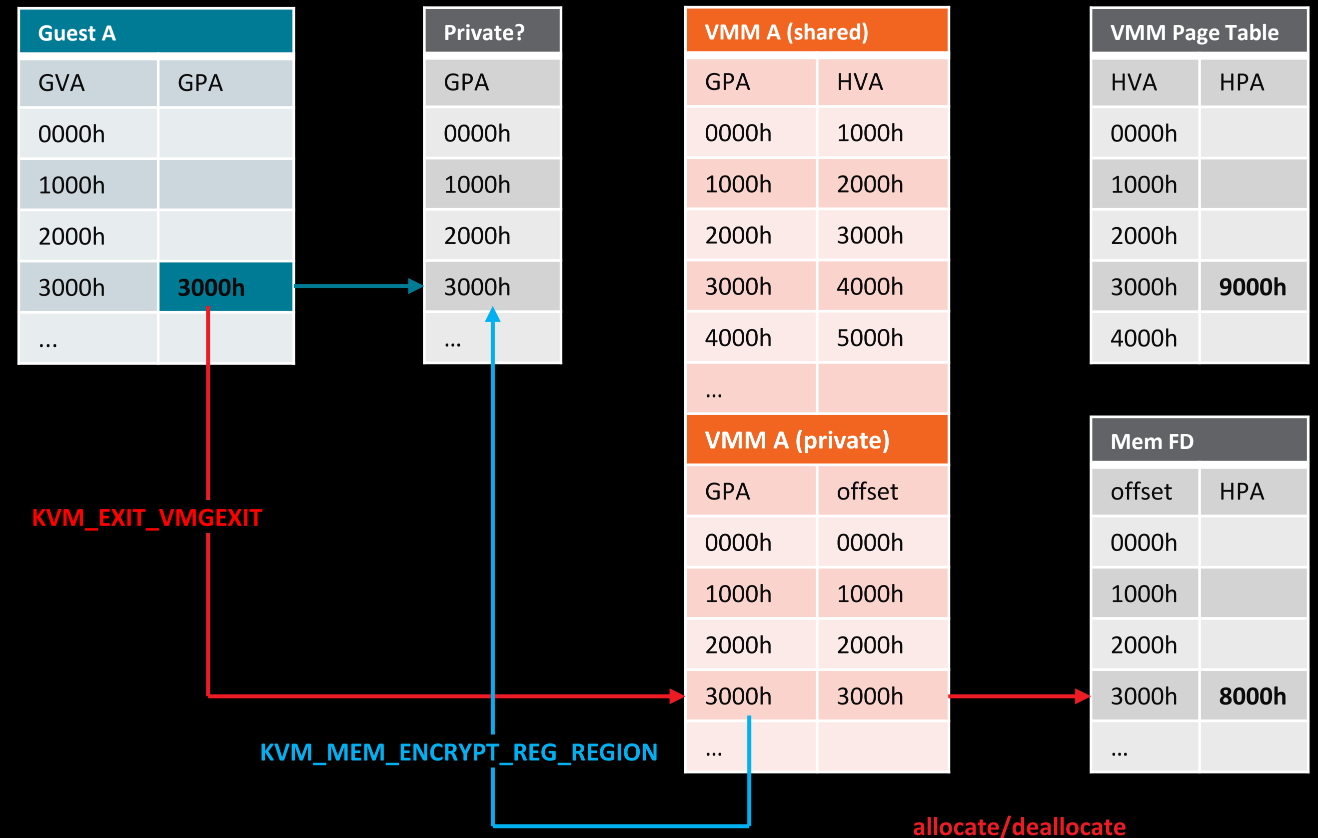


AMD

[Public]

UPM – EXPLICIT CONVERSIONS

- KVM MMU uses an xarray to determine whether to map guest memory from shared/private pool
  - xarray controlled purely by userspace
    - KVM_MEM_ENCRYPT_REG_REGION
    - KVM_MEM_ENCRYPT_UNREG_REGION
- Implicit conversion
  - if C-bit does not match xarray state:
    - KVM_EXIT_MEMORY_FAULT
    - alloc/dealloc private/shared memory
    - VMM converts using REG/UNREG ioctl
- Explicit conversion
  - GHCB page-state change request forwarded to userspace
    - KVM_EXIT_VMGEXIT
    - alloc/dealloc private/shared memory
    - VMM converts using REG/UNREG ioctl



AMD

[Public]

UPM: PROS/CONS

- Pros:
  - Shared infrastructure for managing private guest pages
    - Cross-platform: SNP / TDX, potentially cross-architecture
  - Less chance of guest disruption/exploitation from accessing private memory in userspace
  - Lazy-pinning support
- Cons:
  - More management complexity in VMMs:
    - Allocating/de-allocating private memory
      - Potential for 2X memory usage
        - Lazily-deallocate for performance?
        - Immediately deallocate to reduce memory usage?
    - Handling of new private memslot structure
    - Memory pinning/affinity considerations
  - Performance
    - More exits to userspace, more context switches

AMD

[Public]

HANDLING FOR KERNEL DIRECTMAP

- Writes via 2M directmap mapping will generate RMP violation if it overlaps with private page
- 3 potential approaches:
  - Leave private page mapped, but split the directmap
    - We have set_memory_4k() for this, but no set_memory_2m() currently (solvable?)
  - Remove private page from directmap
    - Basically ends up splitting unless conversion covers whole 2M range
    - Easier to restore 2M mapping (but again, only if conversion covers whole 2M range)
  - Always allocate from private pool with 2M pages
    - No chance for other threads to write into range
    - No splitting needed
    - Feasible?
- Should UPM handle this at all? If so, which approach?

[Public]

GUARDING HOST ACCESSES TO SHARED PAGES AGAINST SHARED->PRIVATE CONVERSIONS

- Host may be accessing shared pages for a number of difference purposes:
  - kvmclock
  - virtio buffers
  - GHCB pages
  - Accesses may be via kernel mappings (e.g. kvm_vcpu_map())
- Ideally:
  - A) If guest erroneously/maliciously flips the page to private, the host should be made aware of this
  - B) If host erroneously/maliciously writes to a page that is now private, the guest should be made aware of this
- SNP (non-UPM) will likely address this situation by using flipping the page back to shared state in the RMP table, this will result in the guest getting a #VC exception if the host did this in error. Provides ideal handling for both A) and B)
- UPM: Separate physical memory pools for shared and private:
  - Case A): helps avoid host crash, but host may not notice unless there's some additional synchronization/invalidation mechanism (not UPM-specific issue, but still an argument in favor of platform-specific handling)
  - Case B): guest won't know host is writing updates to a stale page and silently break, not necessarily corrupting guest memory, but similar end result
  - Also, some archs may not be able to use separate memory pools for shared/private
- Keep this handling platform-specific, or can UPM improve on this somehow?

AMD

[Public]

SCATTER/GATHER SUPPORT FOR KVM_EXIT_MEMORY_FAULT

- SNP does explicit shared/private conversions via GHCB requests
  - Current UPM implementation forwards these to userspace via KVM_EXIT_VMGEXIT
  - Multiple KVM_MEM_ENCRYPT_REG_REGION handled “atomically” by VMM from KVM perspective
- Alternative:
  - Forward these to userspace as KVM_EXIT_MEMORY_FAULT, as with implicit conversions
  - Only handles 1 range at a time, KVM needs to generate multiple KVM_EXIT_MEMORY_FAULTs before completing GHCB request
  - SG list support for KVM_EXIT_MEMORY_FAULT might improve on this
    - Better performance
    - Less complexity in KVM GHCB request handling

AMD

[Public]

Copyright and disclaimer

- ©2022 Advanced Micro Devices, Inc. All rights reserved.
- AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.
- 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 releases, for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. 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.
- THIS INFORMATION IS PROVIDED 'AS IS." 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 NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION

AMD