KVM FORUM

Unmapped Guest Memory

Yu Zhang <yu.c.zhang@intel.com>

THE LINUX FOUNDATION

Legal Disclaimer

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.

Intel and the Intel logo are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

Agenda

- Background
- Unmapping Requirement
- Design Target
- Design Options

Background

Mapping guest memory into host userspace is a common practice in current KVM, to

- Provide emulation(e.g. in KVM/Qemu/vhost-user etc.).
- Facilitate GPA -> HPA translation.

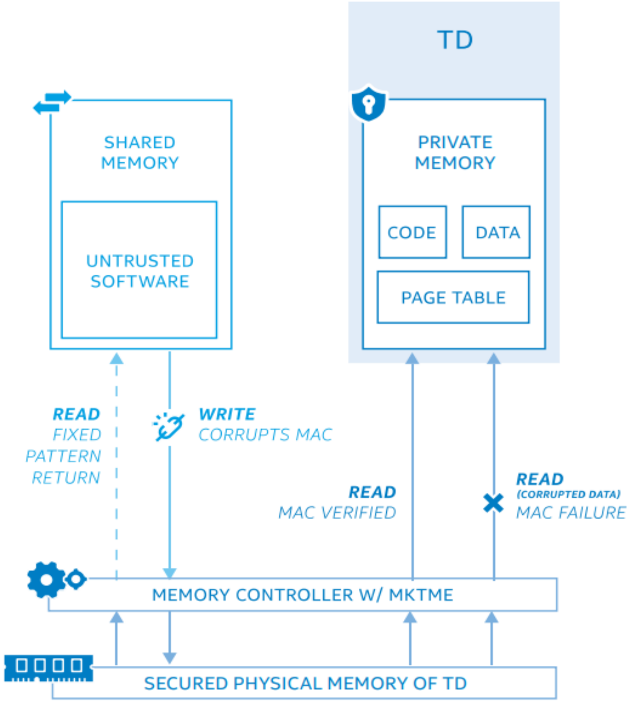
Requirement changes when KVM is no longer inside the TCB:

- Guest private memory shall not be accessible to the host.

Background

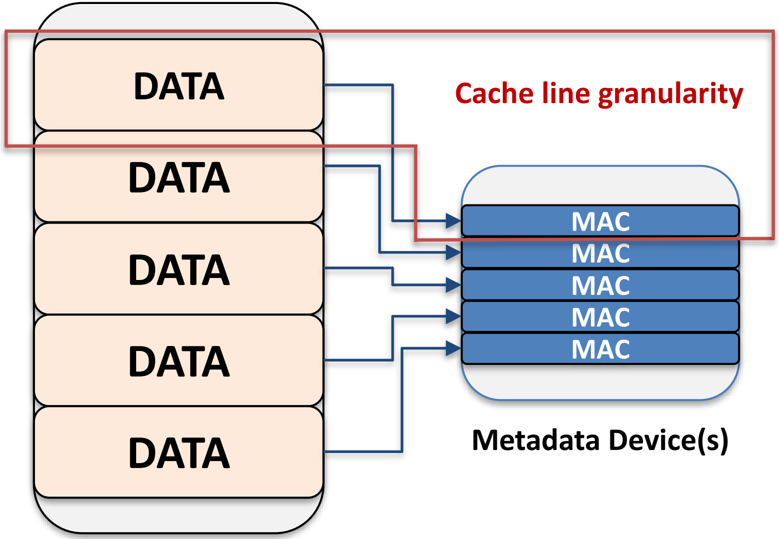
- Intel® TDX uses MKTME engine, to
  - Enable memory encryption.
  - Provide memory integrity protection.

- MKTME with Integrity Architecture offers mechanism to
  - Prevent cyphertext analysis.
  - Prevent data modification without detection.


TD
SHARED MEMORY
UNTRUSTED SOFTWARE
PRIVATE MEMORY
CODE
DATA
PAGE TABLE
READ FIXED PATTERN RETURN
WRITE CORRUPTS MAC
READ MAC VERIFIED
READ (CORRUPTED DATA) MAC FAILURE
MEMORY CONTROLLER W/ MKTME
SECURED PHYSICAL MEMORY OF TD

Background

- MKTME Message Authentication Code(MAC).
  - Associated with each cache line.
  - A 28-bit metadata stored in ECC bits.
  - Generated when a cache line is written to memory.
  - Verified when a cache line is loaded from memory.
- TD-owner bit.
- CPU poisons cache line when memory integrity check fails.



Background

- On subsequent consumption of a poisoned cache line:
  - #MCE
  - Unbreakable shutdown.
- Incorrect or malicious writes from to TD private memory may lead to system crash. ☹
- Solution: map private memory to one specific guest only.
  - No mappings in other guests.
  - No mappings in host userspace.

Unmapping Requirement

No need to map all guest memory in HVA as long as

- GPA -> HPA translation can be done.
- Shared buffer is known to the host.

Private vs. Shared

- Private
  - Guest page tables
  - Guest code pages
  - Guest normal data
- Shared
  - Guest DMA buffers(SWIOTLB buffer & dma_direct_alloc() pages)
  - PV clock

How to unmap & how to get GPA->HPA?

Design Target

About unmapping

- Sharing/unsharing requested only by guest.
  - Guest kernel
  - Virtual BIOS (e.g., TDVF)
- Host page table still managed by Linux MM, not by KVM.
- Kernel direct mapping *can* also be removed.
- Transition between sharing and unsharing.

About GPA-> HPA translation

- 1:1 <ASID, GPA> : HPA association.
- No impact on normal VM mappings.

Option 1 – struct page based

Unmapped Guest Memory

- Guest memory still mmap()ed during VM creation time.
- HVA -> HPA translations blocked, with PFN information kept in host PTE.
- SIGBUS generated to userspace on private memory accesses.

GPA->HPA translation

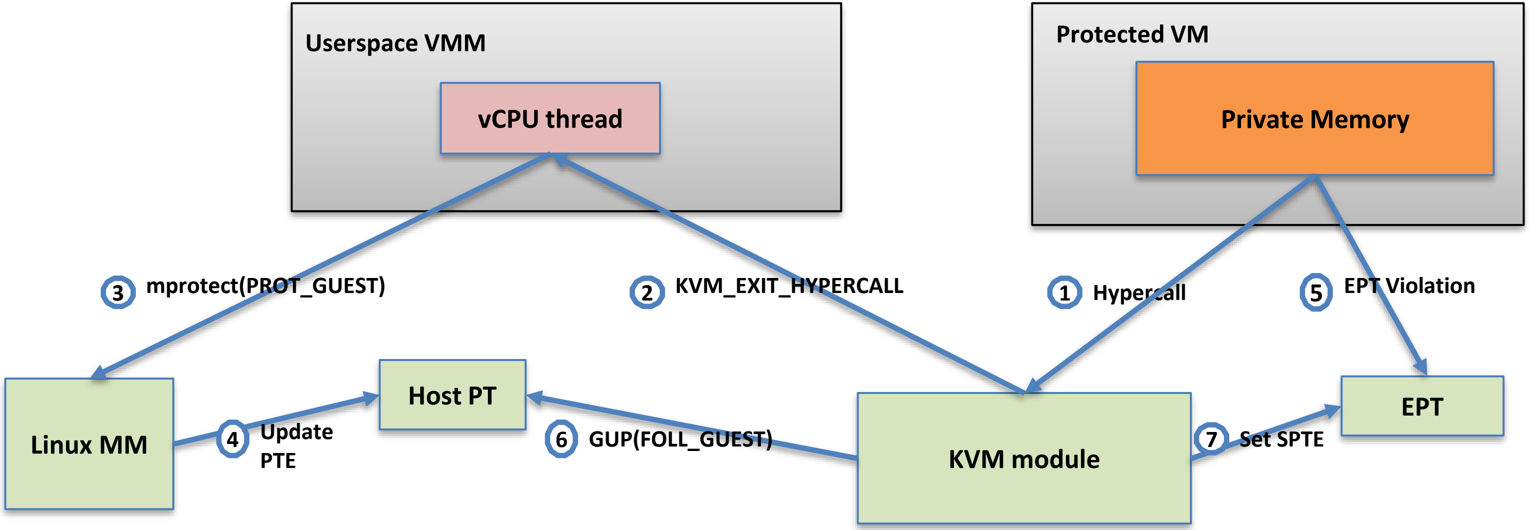
- New GUP flags (e.g., FOLL_ALLOW_POISONED / FOLL_GUEST) to get PFN.
- 1:1 association between <ASID, GPA> and HPA relies on TDX module.
- Does not work for memory that isn't backed by 'struct page’

Option 1 – struct page based

- V1 – HWPOISON [1]
  - Leverages existing flag in struct page: PG_hwpoison.
  - Unmaps HVA in host page table by KVM, with a SWP_HWPOISON entry in PTE.
- V2 – PageGuest [2]
  - Introduces new struct page flag PG_guest.
  - Introduces new mprotect() flags and new VMA flags.
  - Maps/unmaps HVA by Linux MM.

[1] https://lore.kernel.org/kvm/20210402152645.26680-1-kirill.shutemov@linux.intel.com/
[2] https://lore.kernel.org/kvm/20210521123148.a3t4uh4iezm6ax47@box/

Option 1 – struct page based



Option 2 – fd based

Unmapped Guest Memory

- Guest private memory backed by an "enlightened" file descriptor. [3]
- File descriptor is dedicated and private.
  - Not sharable between multiple processes.
  - Not mappable into userspace.
- New fcntl() flags - F_SEAL_GUEST
  - To convert the entire file to guest private memory.
  - To truncate the file size to 0.
- Similar proposal in KVM Forum 2019 (to decouple TDP translation with host page table). [4]

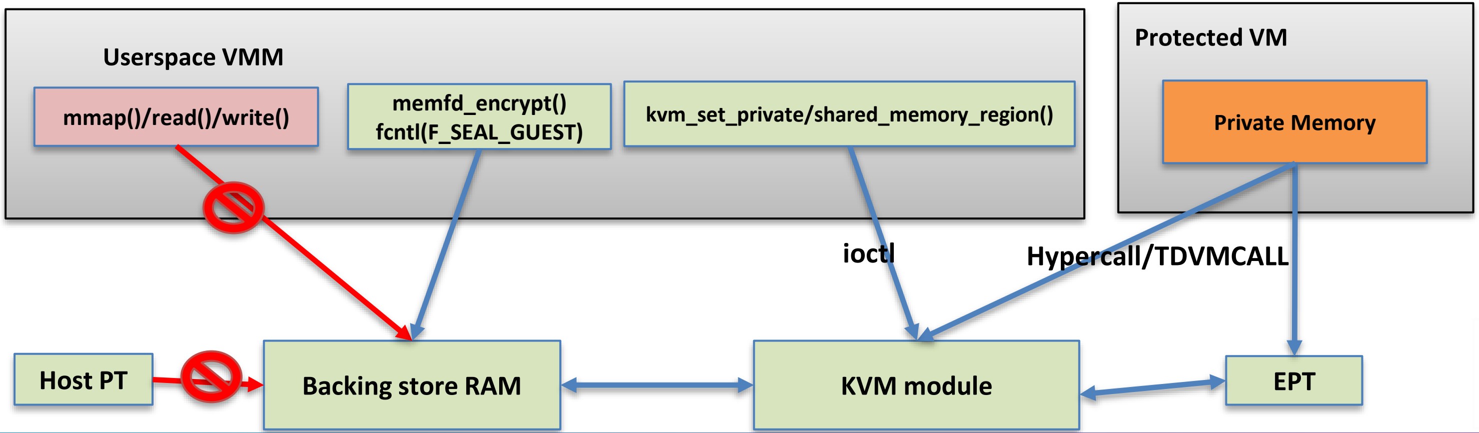
[3] https://lore.kernel.org/lkml/20210824005248.200037-1-seanjc@google.com/

[4] https://static.sched.com/hosted_files/kvmforum2019/48/kvm-forum-vm-memory-mgmt.pdf

Option 2 – fd based

GPA -> HPA translation

- New memslots for guest private memory, possibly a new address space.
- New private ops in memslot offered. E.g.,
  - Ops from backing store
    - gfn_to_pfn() by struct file + offset -> HPA.
    - pfn_mapping_level() for huge page mapping.
  - Ops from KVM to support invalidation/swap/migrate a GFN range.



Option 2 – fd based

Pros

- With fd dedicated to one guest, easy to enforce 1:1 <ASID, GPA> : HPA association.
- Small footprint with much less host page table populated.
- Lower probability of making private memory accessible.
- Easy to support memory not backed with struct page.

Cons

- Significant changes in KVM. E.g.,
  - New ioctls to punch holes in memslots when converting private -> shared.
  - Number of shared memslots could be large
- More reliance on backing store support.
- Non-trivial changes in VFIO interface(if assigned device is desired).

Conclusion

- Unmapped guest memory is not only possible, but also desirable.
- 1:1 <ASID, GPA> : HPA association is feasible, but with cost.
- Let’s make the cost affordable. ☺

Acknowledgement

Sean Christopherson seanjc@google.com

Kirill A. Shutemov kirill.shutemov@linux.intel.com

Isaku Yamahata isaku.yamahata@intel.com

Andy Lutomirski luto@kernel.org

David Hildenbrand david@redhat.com

KVM
FORUM