ACCOUNTING AND PAGE MIGRATION CHALLENGES IN SECURE GUESTS USING FD-BASED PRIVATE MEMORY

Nikunj Dadhania, Michael Roth

KVM FORUM 2023

FD-BASED PRIVATE MEMORY OVERVIEW

- Currently userspace uses malloc()/mmap() to allocate memory, then uses virtual addresses to tell KVM what memory to use to back guest memory
- Development of hypervisor support for various confidential computing technologies drove the need for a different approach to managing confidential guest memory.
- Essentially involves using FD+offset to assign memory instead of userspace virtual addresses in the case of private memory
- Challenges with FD-based private memory
  - Page migration
  - Memory Accounting
  - NUMA

FD-BASED GUEST MEMORY: GUEST_MEMFD

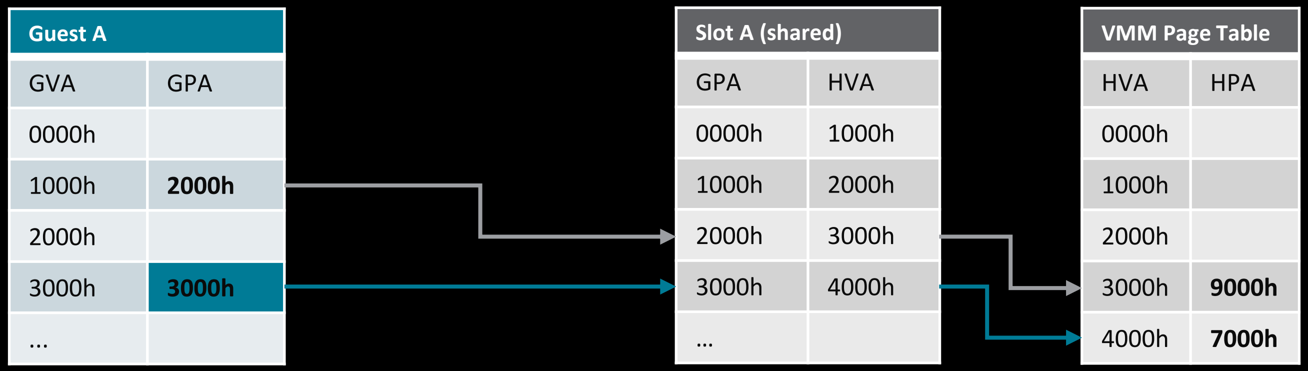
- guest_memfd (gmem)
  - A.K.A. Unmapped Private Memory
  - Previously known as “restrictedmem”, “guardedmem”, “private memfd”
- Provides a way to back private guest memory with pages that can’t be mapped or written to by userspace
  - Provides additional protections against tampering/corrupting guest memory from userspace
  - **Needed** for some platforms where userspace tampering is fatal to the host
- Provides a way to partition shared/private guest memory into separate memory pools
  - **Needed** for some platforms to avoid dealing with things like shared->private state transitions while host is attempting to access a shared page (virtio/DMA buffers, GHCB pages for SEV-SNP)
- How does it work?

AMD

NORMAL 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 guest_memfd
    - 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

#NPF: GPA->HPA lookup (normal memslot)



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 guest_memfd
    - 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

#NPF: GPA->HPA lookup (private memslot)

Guest A

GVA	GPA
0000h	
1000h	**2000h**
2000h	
3000h	**3000h**
…	

Private?

GPA
0000h
1000h
2000h
3000h
…

Slot A (shared)

GPA	HVA
0000h	1000h
1000h	2000h
2000h	3000h
3000h	4000h
…	

VMM Page Table

HVA	HPA
0000h	
1000h	
2000h	
3000h	**9000h**
4000h	

Slot A (private)

GPA	offset
0000h	0000h
1000h	1000h
2000h	2000h
3000h	3000h
…	

gmem FD

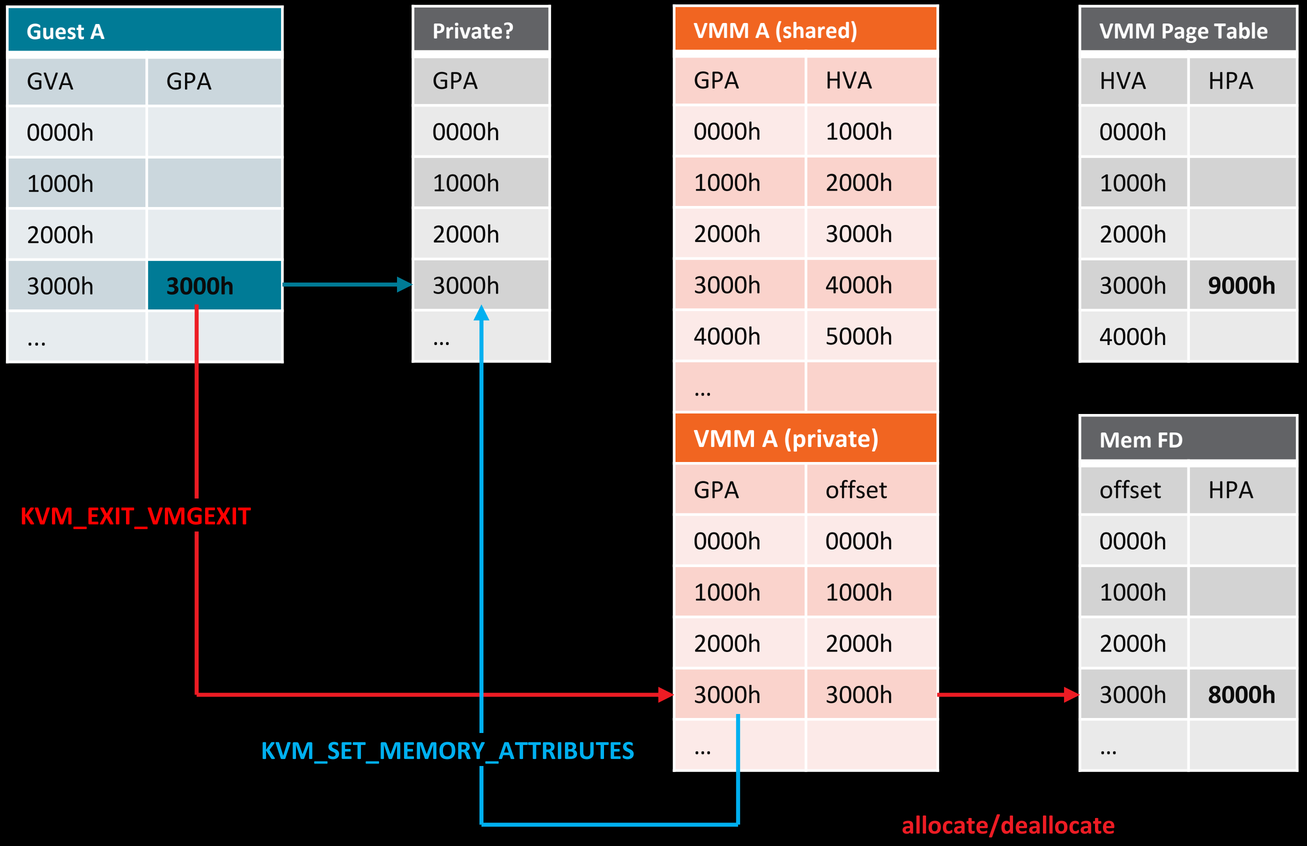
offset	HPA
0000h	
1000h	
2000h	
3000h	**8000h**
…	

AMD

USING FD-BASED MEMORY FOR GUESTS

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

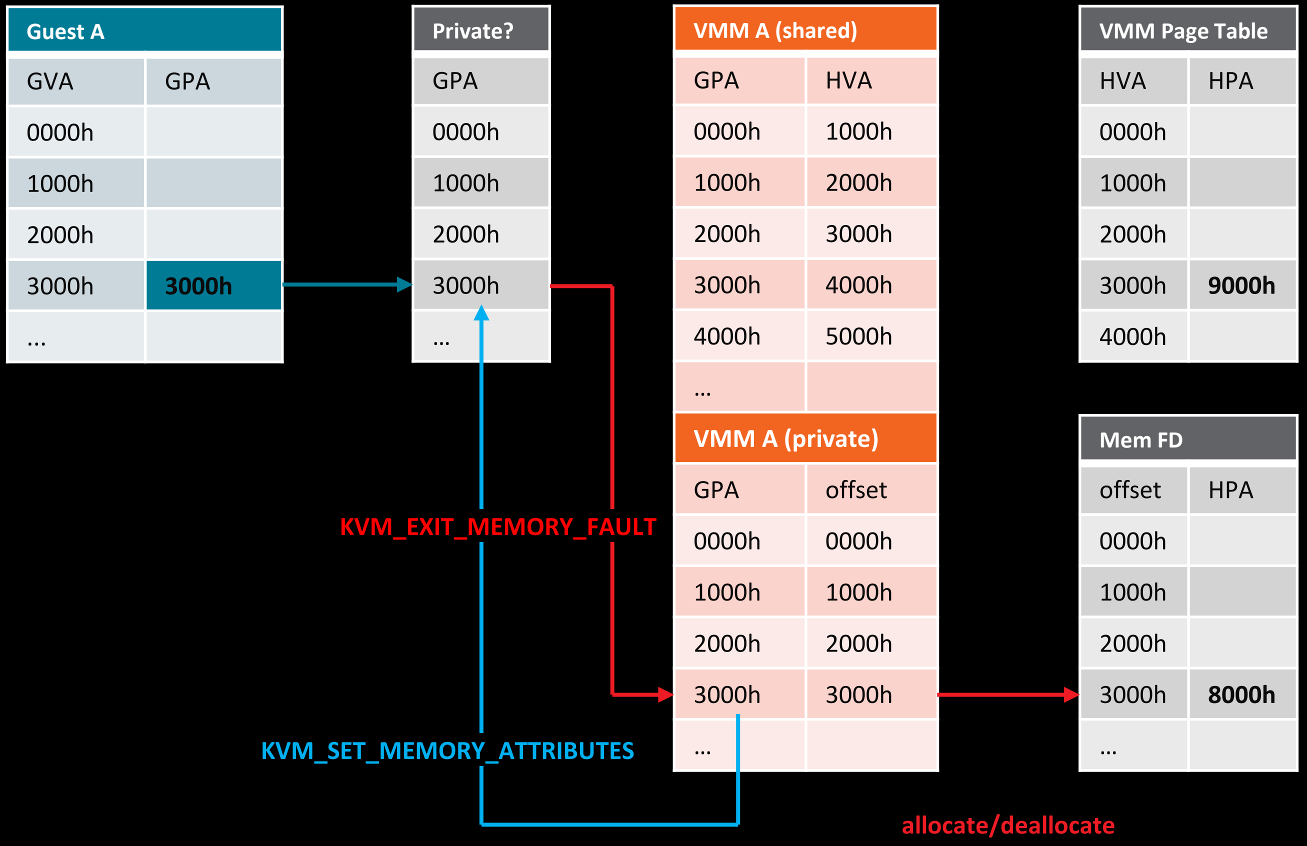
#NPF: GPA->HPA lookup/conversion (restricted memslot)



USING FD-BASED MEMORY FOR GUESTS

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

#NPF: GPA->HPA lookup/conversion (restricted memslot)



PAGE MIGRATION

- gmem does not currently support page migration, likely a follow-up feature
- Will be wanted eventually
  - Memory compaction/defragmentation
  - NUMA rebalancing
  - Cgroup movements
  - memory offlining/unplug
  - migratepages command or syscalls like move_pages and migrate_pages
- Number of issues need to be addressed to get these working for gmem / confidential guests
  - Let’s look at some of these issues in the context of memory compaction

PAGE MIGRATION FOR MEMORY COMPACTION

- kcompactd thread runs periodically to migrate pages from sparse areas to denser ones
  - Helps reduce memory fragmentation to avoid failures for contiguous allocations
  - Improves availability of THPs
- As with other subsystems, relies on migrate_pages() interface to handle the migrations

PAGE MIGRATION FOR MEMORY COMPACTION

- migrate_pages(from_list, new_page_fn, put_page_fn, mode, reason, …)
  - from_list: list of folios/pages to migrate
  - new_page_fn: compaction_alloc(), scans for suitable destinations pages starting at end of zone
    - Favors low-order pages, and pages from movable zones to avoid fragmenting non-movable zones where migrations are disallowed/discouraged (e.g., DMA memory)
  - put_page_fn: compaction_free(), on failure, puts destination pages back on freelist
  - mode: MIGRATE_ASYNC, MIGRATE_SYNC, …
  - reason: MR_COMPACTION
    - Other subsystems might use MR_MEMORY_HOTPLUG, MR_MEMPOLICY_MBIND, etc.

AMD

PAGE MIGRATION FOR MEMORY COMPACTION

- migrate_pages(…)
  - migrate_folio_unmap(src, &dst, …)
    - Use src folio's rmap to find VMAs that map the page into userspace
    - Unmap and issue MMU notifier events so subscribers like KVM can unmap from guest TDP/NPT/EPT
    - **But gmem is never mapped… so no KVM MMU invalidations are issued. Use-after-free!**
  - migrate_folio_move(src, &dst, …)
    - Use gmem's migrate_folio callback to handle copying src to dst
    - **But private memory generally cannot be migrated without hardware/firmware support.**
- Possible solution
  - gmem's migrate_folio callback can provide hooks to handle platform-specific requirements (e.g., SNP_PAGE_MOVE firmware commands for SEV-SNP)
  - gmem is owned by KVM, so gmem migrate_folio callback can handle KVM MMU invalidations directly
- Acceptable for compaction maybe, but other page migration users like cgroups/NUMA rely on VMA-based memory accounting to make migration decisions…

AMD

MEMORY ACCOUNTING

- gmem FD allocations are currently counted as usage page cache allocations
  - Not accounted to current process
- Adversely impacts accounting for a number of areas
  - Cgroups
  - General process limits
  - NUMA
- For example…

MEMORY ACCOUNTING ISSUES

- Start SNP guest 40G memory with memory interleave between Node2 and Node3

```
numactl -i 2,3 ./bootg_snp.sh
```

- Incorrect process resident memory is reported
- Although NUMA allocation came from Node2 and Node3, does not get attributed to QEMU process
- Uses process mempolicy for proper node allocation

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 4279 root      20   0   10.4g  85332  49308 S 100.3   0.0   0:27.66 qemu-system-x86
```

```
Every 1.0s: sudo numastat  -m -p qemu-system-x86 | egrep -i "qemu|PID|Node|Filepage"

Per-node process memory usage (in MBs)
PID                               Node 0          Node 1          Node 2          Node 3           Total
31921 (qemu-system-x86)            10.61            6.72           46.96           25.50           89.79
Per-node system memory usage (in MBs):
                          Node 0          Node 1          Node 2          Node 3           Total
FilePages                2172.80         3488.22        21500.98        22480.20        49642.20
```

MEMORY ACCOUNTING ISSUES

- /proc/<pid>/numa_maps
- /proc/<pid>/smaps
- Uses VMAs to populate memory usage per NUMA node
- /memfd:memory-backend-memfd-private missing

```
↪ grep memfd /proc/6195/numa_maps
7f291aa00000 interleave:2-3 file=/memfd:rom-backend-memfd-shared\040(deleted)
7f291ae00000 interleave:2-3 file=/memfd:rom-backend-memfd-shared\040(deleted) dirty=32 active=0 N2=16 N3=16 kernelpagesize_kB=4
7f291bc00000 interleave:2-3 file=/memfd:memory-backend-memfd-shared\040(deleted) anon=8 dirty=8 mapped=38 active=7 N2=21 N3=17 kernelpagesize_kB=4
7f2ba0200000 interleave:2-3 file=/memfd:rom-backend-memfd-shared\040(deleted) dirty=892 active=0 N2=446 N3=446 kernelpagesize_kB=4
```

```
↪ grep memfd /proc/6195/smaps
7f291aa00000-7f291aa20000 rw-s 00000000 00:01 4120        /memfd:rom-backend-memfd-shared (deleted)
7f291ae00000-7f291ae20000 rw-s 00000000 00:01 4118        /memfd:rom-backend-memfd-shared (deleted)
7f291bc00000-7f2b9bc00000 rw-p 00000000 00:01 4114        /memfd:memory-backend-memfd-shared (deleted)
7f2ba0200000-7f2ba057c000 rw-s 00000000 00:01 4116        /memfd:rom-backend-memfd-shared (deleted)
```

MEMORY ACCOUNTING – POTENTIAL SOLUTIONS

- Memory accounting relies heavily on VMAs, as does migration
  - Give it what it wants?
- Use shadow/invisible VMAs for guest-mapped gmem ranges
  - Need to ensure mappings don't get put in process page tables, or aren't actually visible by hardware
    - Maybe some architectures don't provide such a thing
    - Could provide alternative hooks for "shadow" VMAs for handling translations throughout kernel
      - Wire those lookups directly up to the TDP?
  - Just having a VMA isn't enough, some accounting happens via page fault handler
    - Duplicate that accounting when mapping gmem pages into TDP? During initial allocation?
- Alternative: implement a completely separate alternative to using VMAs for accounting?
  - Not necessarily better
- Needs a lot more discussion/investigation

AMD

SUMMARY

- gmem/UPM provides the critical framework needed to finally enable confidential computing for KVM, but many gaps remain WRT page migration and memory accounting
- Current implementation will likely be acceptable for many users who value security above all, but eventually we will need to close some of these gaps
- Potential solutions exist, but no clear/simple path yet
- Will need input from the community and the memory experts to get there

- **Thanks!**

Copyright and disclaimer

AMD

OTHER ISSUES – NUMA VIA MBIND()

- Basic NUMA is possible via numactl / set_mempolicy()
  - but process-wide policies aren't enough, and QEMU generally doesn’t rely on them for NUMA

```
qemu \
  -numa node,nodeid=0,cpus=0-1,memdev=mem0 \
  -object memory-backend-memfd-private,id=mem0,policy=bind,host-nodes=0 \
  -numa node,nodeid=1,cpus=2-3,memdev=mem1 \
  -object memory-backend-memfd-private,id=mem1,policy=bind,host-nodes=1
```

- Each memory backend instance will use mbind() to set policy for that particular memory range
  - **but mbind() needs a virtual address, and the gmem FD can’t be mmap()’d, only the FD representing shared pool gets the mmap()/mbind()**
- Potential solutions
  - Implement a new fbind() ioctl?
  - Have KVM duplicate the mempolicy for shared pool onto the private/gmem pool underneath the covers?