DEBUG support for Confidential Guest

Ashish Kalra
SMTS Software System Design Engineer, AMD
DEBUG support for Confidential Guests

- Encrypted guest has its memory and/or register context encrypted by Vendor specific technology (AMD SEV / Intel TDX).

- Encrypted guest memory breaks down QEMU’s built-in debugging features because it cannot do direct guest memory access (memcpy() from HVA).

- Introduce a basic framework and common x86 implementation to handle encrypted guest memory reading/writing to support QEMU’s built-in debugging features like monitor xp command and gdbstub.
DEBUG support for Confidential Guests

- QEMU debug support for Confidential VM/guests
- Debug requires access to guests pages which are encrypted for example when SEV is enabled
- SEV APIs
  - DBG_DECRYPT
  - DBG_ENCRYPT

for decryption/encryption of guest pages if guest policy allows for debugging
Extension of MemTxAttrs

- Extend the MemTx Attrs to include a “debug” flag to indicate that the operation is triggered by a debugger.

```c
typedef struct MemTxAttrs {
...
...  /* Memory access requested from the debugger */
+ unsigned int debug : 1;
} MemTxAttrs;
```
Memory Debug Ops

- Introduce new “MemoryDebugOps” which hook into guest virtual & physical memory debug interfaces such as cpu_memory_rw_debug() to allow vendor specific assist/hooks for debug access to guest memory.

typedef struct MemoryDebugOps {
    hwaddr (*translate) (CPUState *.., target_ulong addr, MemTxAttrs *attrs);
    MemTxResult (*read) (AddressSpace *as, hwaddr phys_addr, MemTxAttrs attrs, void *buf, hwaddr len);
    MemTxResult (*write) (AddressSpace *as, hwaddr phys_addr, MemTxAttrs attrs, const void *buf, hwaddr len);
    uint64_t (*pte_mask) (void);
} MemoryDebugOps;
These ops would be used by cpu_memory_rw_debug() and would default to:

```c
static const MemoryDebugOps default_debug_ops = {
    .translate = cpu_get_phys_attrs_debug,
    .read = address_space_read,
    .write = address_space_write_rom,
    .pte_mask = address_space_pte_mask,
};
```

```c
cpu_memory_rw_debug (cpu, addr) {
    ...

    /* Set debug attrs to indicate memory access is from debugger */
    attrs.debug = 1;

    if (is_write) {
        res = address_space_write_rom (phys_addr, attrs);
        res = debug_ops->write (phys_addr, attrs);
    } else {
        res = address_space_read (phys_addr, attrs);
        res = debug_ops->read (phys_addr, attrs);
    }
}
```

If (is_write) {
- `res = address_space_write_rom (phys_addr, attrs);
+ res = debug_ops->write (phys_addr, attrs);
}
```
Additionally, introduce new MemoryRegion debug ops.

Extend ‘struct MemoryRegion’ to include new callbacks that can be used to override use of memcpy() with something appropriate for SEV/confidential memory guests.

/* MemoryRegion RAM debug callbacks*/

typedef struct MemoryRegionRAMReadWriteOps MemoryRegionRAMReadWriteOps;

struct MemoryRegionRAMReadWriteOps {
    /* write data into guest memory*/
    int (*write) (uint8_t *dest, ..src, ..len, ..attrs);

    /* read data from guest memory */
    int (*read) (..dest, ..src, ..len, ..attrs);
};

/** MemoryRegion ...

... */

struct MemoryRegion {
    ...

    + const MemoryRegionRAMReadWriteOps* ram_debug_ops;

Debug API flow for SEV guests

cpu_memory_rw_debug()
  debug_ops->read(..)
  sev_address_space_read_debug(..)
  /* invoke address_space_rw_debug helpers */
  address_space_read_debug (as, addr, attrs, ptr, len),
  ...
  mr = address_space_translate (..);
  ...
  //RAM case
  ram_ptr = ... ;
  if (attrs.debug && mr->ram_debug_ops)
    mr->ram_debug_ops->read (buf, ram_ptr, l, attrs):
  else
    memcpy (buf, ram_ptr, l);
continued.....

sev_mem_read(..)

sev_dbg_enc_dec(..)

sev_ioctl(sev_fd,

        write ? KVM_SEV_DBG_ENCRYPT : KVM_SEV_DBG_DECRYPT),

        &dbg, ...);
Add debug versions of physical memory read & write APIs
-cpu_physical_memory_read_debug
-cpu_physical_memory_write_debug
-cpu_physical_memory_rw_debug
-ldl_phys_debug
-ldq_phys_debug

-these internally invoke MemoryDebugOps

-use above debug APIs when accessing guest memory.

for example:

```
tlb_info_32(..)
{
    for (l1=0; l1<1024; l1++) {
        - cpu_physical_memory_read (pgd+l1*4, &pde, 4);
        + cpu_physical_memory_read_debug (pgd+l1*4, &pde, 4);
    }
```
In SEV-enabled guest, the pte entry will have c-bit set, need to clear the c-bit when walking the page table, to ensure that proper page address translation occurs & with c-bit reset, the true physical address is retrieved.

tlb_info_pae32(..)
{
    uint64_t me_mask;
    me_mask = cpu_physical_memory_pte_mask_debug();
    /* debug_ops → pte_mask(); */
    ...
    pdp_addr==...
    pdp_addr &= me_mask;
    for (I1=0; I1<4; I1++) {
        cpu_physical_memory_rw_debug (pdp_addr + I1*8, &pdpe, 8);
        pdpe = le64_to_cpu (pdpe);
        pdpe = le64_to_cpu(pdpe & me_mask);
    } /*similarly for pde & pte's */
Currently (*translate) in MemoryDebugOps is not used....

sev_cpu_get Phys_attrs_debug(..)
{
  .
}
this is invoked by directly overriding cpu class page table walker :

sev_set_debug_ops_cpu_state (..*handle, CPUState *cs)
{
  CPUClass *cc;

  /*If policy does not allow debug then no need to register ops*/
  If (s→ policy & SEV_POLICY_NODBG) {
    return;
  }
  cc = CPU_GET_CLASS(cs);

  /* override guest mmu lookup/page table walker with SEV specific callback to handle encrypted memory*/
  cc→ get_phys_page_attrs_debug = sev_cpu_get_phys_atrrs_debug();

  address_space_set_debug_ops (& sev_debug_ops);
Summary: Relationship diagram of the APIs & Interface

- x86_cpu_get_physAttrs_debug()
- cpu_physical_memory_rw_debug()
- (x86_) Idl_phys_debug()
- (x86_) Idq_phys_debug()
- address_space_read()
- address_space_write_rom()
- address_space_read_debug()
- address_space_write_rom_debug()
- MemoryDebugOps *debug_ops
- MemoryRegionRAMReadWriteOps *ram_debug_ops
- kvm_encrypted_guest_read_memory() /
  kvm_encrypted_guest_write_memory()
- kvm_vm_ioctl() / sev_ioctl()
Limitations:

- Encrypted Register State Access and Debugging
  
  Device Access / Debugging (device emulation issues)
  
  Guest awareness?
  
  Virtual INT1/INT3 injection?
  
  VC# handler and GHCB extensions?