Secure Encrypted Virtualization (SEV)

Overview

Secure Encrypted Virtualization (SEV) is a feature found on AMD processors.

SEV is an extension to the AMD-V architecture which supports running virtual machines (VMs) under the control of a hypervisor. When enabled, the memory contents of a VM will be transparently encrypted with a key unique to that VM.

The hypervisor can determine the SEV support through the CPUID instruction. The CPUID function 0x8000001f reports information related to SEV:

```
0x8000001f[eax]:
                Bit[1]  indicates support for SEV
    ...
          [ecx]:
                Bits[31:0]  Number of encrypted guests supported simultaneously
```

If support for SEV is present, MSR 0xc001_0010 (MSR_AMD64_SYSCFG) and MSR 0xc001_0015 (MSR_K7_HWCR) can be used to determine if it can be enabled:

```
0xc001_0010:
        Bit[23]    1 = memory encryption can be enabled
                   0 = memory encryption can not be enabled

0xc001_0015:
        Bit[0]     1 = memory encryption can be enabled
                   0 = memory encryption can not be enabled
```

When SEV support is available, it can be enabled in a specific VM by setting the SEV bit before executing VMRUN.:

```
VMCB[0x90]:
        Bit[1]      1 = SEV is enabled
                    0 = SEV is disabled
```

SEV hardware uses ASIDs to associate a memory encryption key with a VM. Hence, the ASID for the SEV-enabled guests must be from 1 to a maximum value defined in the CPUID 0x8000001f[ecx] field.

SEV Key Management

The SEV guest key management is handled by a separate processor called the AMD Secure Processor (AMD-SP). Firmware running inside the AMD-SP provides a secure key management interface to perform common hypervisor activities such as encrypting bootstrap code, snapshot, migrating and debugging the guest. For more information, see the SEV Key Management spec [api-spec]

The main ioctl to access SEV is KVM_MEMORY_ENCRYPT_OP. If the argument to KVM_MEMORY_ENCRYPT_OP is NULL, the ioctl returns 0 if SEV is enabled and `ENOTTY` if it is disabled (on some older versions of Linux, the ioctl runs normally even with a NULL argument, and therefore will likely return ``EFAULT``). If non-NULL, the argument to KVM_MEMORY_ENCRYPT_OP must be a struct kvm_sev_cmd:

```
struct kvm_sev_cmd {
        __u32 id;
        __u64 data;
        __u32 error;
        __u32 sev_fd;
};
```

The `id` field contains the subcommand, and the `data` field points to another struct containing arguments specific to command. The `sev_fd` should point to a file descriptor that is opened on the `/dev/sev` device, if needed (see individual commands).

On output, `error` is zero on success, or an error code. Error codes are defined in `<linux/psp-dev.h>`.

KVM implements the following commands to support common lifecycle events of SEV guests, such as launching, running, snapshotting, migrating and decommissioning.

1. KVM_SEV_INIT

The KVM_SEV_INIT command is used by the hypervisor to initialize the SEV platform context. In a typical workflow, this command should be the first command issued.

The firmware can be initialized either by using its own non-volatile storage or the OS can manage the NV storage for the firmware using the module parameter `init_ex_path`. The file specified by `init_ex_path` must exist. To create a new NV storage file allocate the file with 32KB bytes of 0xFF as required by the SEV spec.

Returns: 0 on success, -negative on error

2. KVM_SEV_LAUNCH_START

The KVM_SEV_LAUNCH_START command is used for creating the memory encryption context. To create the encryption context, user must provide a guest policy, the owner's public Diffie-Hellman (PDH) key and session information.

Parameters: struct kvm_sev_launch_start (in/out)

Returns: 0 on success, -negative on error

```
struct kvm_sev_launch_start {
        __u32 handle;           /* if zero then firmware creates a new handle */
        __u32 policy;           /* guest's policy */

        __u64 dh_uaddr;         /* userspace address pointing to the guest owner's PDH key */
        __u32 dh_len;

        __u64 session_addr;     /* userspace address which points to the guest session information */
        __u32 session_len;
};
```

On success, the 'handle' field contains a new handle and on error, a negative value.

KVM_SEV_LAUNCH_START requires the `sev_fd` field to be valid.

For more details, see SEV spec Section 6.2.

3. KVM_SEV_LAUNCH_UPDATE_DATA

The KVM_SEV_LAUNCH_UPDATE_DATA is used for encrypting a memory region. It also calculates a measurement of the memory contents. The measurement is a signature of the memory contents that can be sent to the guest owner as an attestation that the memory was encrypted correctly by the firmware.

Parameters (in): struct kvm_sev_launch_update_data

Returns: 0 on success, -negative on error

```
struct kvm_sev_launch_update {
        __u64 uaddr;    /* userspace address to be encrypted (must be 16-byte aligned) */
        __u32 len;      /* length of the data to be encrypted (must be 16-byte aligned) */
};
```

For more details, see SEV spec Section 6.3.

4. KVM_SEV_LAUNCH_MEASURE

The KVM_SEV_LAUNCH_MEASURE command is used to retrieve the measurement of the data encrypted by the KVM_SEV_LAUNCH_UPDATE_DATA command. The guest owner may wait to provide the guest with confidential information until it can verify the measurement. Since the guest owner knows the initial contents of the guest at boot, the measurement can be verified by comparing it to what the guest owner expects.

If len is zero on entry, the measurement blob length is written to len and uaddr is unused.

Parameters (in): struct kvm_sev_launch_measure

Returns: 0 on success, -negative on error

```
struct kvm_sev_launch_measure {
        __u64 uaddr;    /* where to copy the measurement */
        __u32 len;      /* length of measurement blob */
};
```

For more details on the measurement verification flow, see SEV spec Section 6.4.

5. KVM_SEV_LAUNCH_FINISH

After completion of the launch flow, the KVM_SEV_LAUNCH_FINISH command can be issued to make the guest ready for the execution.

Returns: 0 on success, -negative on error

6. KVM_SEV_GUEST_STATUS

The KVM_SEV_GUEST_STATUS command is used to retrieve status information about a SEV-enabled guest.

Parameters (out): struct kvm_sev_guest_status

Returns: 0 on success, -negative on error

```
struct kvm_sev_guest_status {
        __u32 handle;   /* guest handle */
        __u32 policy;   /* guest policy */
```

```
        __u8 state;     /* guest state (see enum below) */
};
```

SEV guest state:

```
enum {
SEV_STATE_INVALID = 0;
SEV_STATE_LAUNCHING,    /* guest is currently being launched */
SEV_STATE_SECRET,       /* guest is being launched and ready to accept the ciphertext data */
SEV_STATE_RUNNING,      /* guest is fully launched and running */
SEV_STATE_RECEIVING,    /* guest is being migrated in from another SEV machine */
SEV_STATE_SENDING       /* guest is getting migrated out to another SEV machine */
};
```

7. KVM_SEV_DBG_DECRYPT

The KVM_SEV_DEBUG_DECRYPT command can be used by the hypervisor to request the firmware to decrypt the data at the given memory region.

Parameters (in): struct kvm_sev_dbg

Returns: 0 on success, -negative on error

```
struct kvm_sev_dbg {
        __u64 src_uaddr;        /* userspace address of data to decrypt */
        __u64 dst_uaddr;        /* userspace address of destination */
        __u32 len;              /* length of memory region to decrypt */
};
```

The command returns an error if the guest policy does not allow debugging.

8. KVM_SEV_DBG_ENCRYPT

The KVM_SEV_DEBUG_ENCRYPT command can be used by the hypervisor to request the firmware to encrypt the data at the given memory region.

Parameters (in): struct kvm_sev_dbg

Returns: 0 on success, -negative on error

```
struct kvm_sev_dbg {
        __u64 src_uaddr;        /* userspace address of data to encrypt */
        __u64 dst_uaddr;        /* userspace address of destination */
        __u32 len;              /* length of memory region to encrypt */
};
```

The command returns an error if the guest policy does not allow debugging.

9. KVM_SEV_LAUNCH_SECRET

The KVM_SEV_LAUNCH_SECRET command can be used by the hypervisor to inject secret data after the measurement has been validated by the guest owner.

Parameters (in): struct kvm_sev_launch_secret

Returns: 0 on success, -negative on error

```
struct kvm_sev_launch_secret {
        __u64 hdr_uaddr;        /* userspace address containing the packet header */
        __u32 hdr_len;

        __u64 guest_uaddr;      /* the guest memory region where the secret should be injected */
        __u32 guest_len;

        __u64 trans_uaddr;      /* the hypervisor memory region which contains the secret */
        __u32 trans_len;
};
```

10. KVM_SEV_GET_ATTESTATION_REPORT

The KVM_SEV_GET_ATTESTATION_REPORT command can be used by the hypervisor to query the attestation report containing the SHA-256 digest of the guest memory and VMSA passed through the KVM_SEV_LAUNCH commands and signed with the PEK. The digest returned by the command should match the digest used by the guest owner with the KVM_SEV_LAUNCH_MEASURE.

If len is zero on entry, the measurement blob length is written to len and uaddr is unused.

Parameters (in): struct kvm_sev_attestation

Returns: 0 on success, -negative on error

```
struct kvm_sev_attestation_report {
        __u8 mnonce[16];        /* A random mnonce that will be placed in the report */

        __u64 uaddr;            /* userspace address where the report should be copied */
        __u32 len;
};
```

11. KVM_SEV_SEND_START

The KVM_SEV_SEND_START command can be used by the hypervisor to create an outgoing guest encryption context.

If session_len is zero on entry, the length of the guest session information is written to session_len and all other fields are not used.

Parameters (in): struct kvm_sev_send_start

Returns: 0 on success, -negative on error

```
struct kvm_sev_send_start {
        __u32 policy;                /* guest policy */

        __u64 pdh_cert_uaddr;        /* platform Diffie-Hellman certificate */
        __u32 pdh_cert_len;

        __u64 plat_certs_uaddr;       /* platform certificate chain */
        __u32 plat_certs_len;

        __u64 amd_certs_uaddr;       /* AMD certificate */
        __u32 amd_certs_len;

        __u64 session_uaddr;         /* Guest session information */
        __u32 session_len;
};
```

12. KVM_SEV_SEND_UPDATE_DATA

The KVM_SEV_SEND_UPDATE_DATA command can be used by the hypervisor to encrypt the outgoing guest memory region with the encryption context creating using KVM_SEV_SEND_START.

If hdr_len or trans_len are zero on entry, the length of the packet header and transport region are written to hdr_len and trans_len respectively, and all other fields are not used.

Parameters (in): struct kvm_sev_send_update_data

Returns: 0 on success, -negative on error

```
struct kvm_sev_launch_send_update_data {
        __u64 hdr_uaddr;        /* userspace address containing the packet header */
        __u32 hdr_len;

        __u64 guest_uaddr;      /* the source memory region to be encrypted */
        __u32 guest_len;

        __u64 trans_uaddr;      /* the destination memory region  */
        __u32 trans_len;
};
```

13. KVM_SEV_SEND_FINISH

After completion of the migration flow, the KVM_SEV_SEND_FINISH command can be issued by the hypervisor to delete the encryption context.

Returns: 0 on success, -negative on error

14. KVM_SEV_SEND_CANCEL

After completion of SEND_START, but before SEND_FINISH, the source VMM can issue the SEND_CANCEL command to stop a migration. This is necessary so that a cancelled migration can restart with a new target later.

Returns: 0 on success, -negative on error

15. KVM_SEV_RECEIVE_START

The KVM_SEV_RECEIVE_START command is used for creating the memory encryption context for an incoming SEV guest. To create the encryption context, the user must provide a guest policy, the platform public Diffie-Hellman (PDH) key and session information.

Parameters: struct kvm_sev_receive_start (in/out)

Returns: 0 on success, -negative on error

```
struct kvm_sev_receive_start {
```

```
        __u32 handle;           /* if zero then firmware creates a new handle */
        __u32 policy;           /* guest's policy */

        __u64 pdh_uaddr;        /* userspace address pointing to the PDH key */
        __u32 pdh_len;

        __u64 session_uaddr;    /* userspace address which points to the guest session information */
        __u32 session_len;
};
```

On success, the 'handle' field contains a new handle and on error, a negative value.

For more details, see SEV spec Section 6.12.

16. KVM_SEV_RECEIVE_UPDATE_DATA

The KVM_SEV_RECEIVE_UPDATE_DATA command can be used by the hypervisor to copy the incoming buffers into the guest memory region with encryption context created during the KVM_SEV_RECEIVE_START.

Parameters (in): struct kvm_sev_receive_update_data

Returns: 0 on success, -negative on error

```
struct kvm_sev_launch_receive_update_data {
        __u64 hdr_uaddr;        /* userspace address containing the packet header */
        __u32 hdr_len;

        __u64 guest_uaddr;      /* the destination guest memory region */
        __u32 guest_len;

        __u64 trans_uaddr;      /* the incoming buffer memory region  */
        __u32 trans_len;
};
```

17. KVM_SEV_RECEIVE_FINISH

After completion of the migration flow, the KVM_SEV_RECEIVE_FINISH command can be issued by the hypervisor to make the guest ready for execution.

Returns: 0 on success, -negative on error

References

See [white-paper], [api-spec], [amd-apm] and [kvm-forum] for more info.

[white-paper] http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf

[api-spec] (*1*,*2*) https://support.amd.com/TechDocs/55766_SEV-KM_API_Specification.pdf

[amd-apm] https://support.amd.com/TechDocs/24593.pdf (section 15.34)

[kvm-forum] https://www.linux-kvm.org/images/7/74/02x08A-Thomas_Lendacky-AMDs_Virtualizatoin_Memory_Encryption_Technology.pdf