SUSE

COCONUT-SVSM on KVM: Progress, Plans and Challenges

Jörg Rödel & Roy Hopkins

KVM Forum 2024

COCONUT-SVSM Progress

Progress

General

- Many enhancements towards idiomatic Rust
  - Safety improvements
  - PerCPU Borrow Checking
- Test improvements
  - Test-in-SVSM - Run tests in SVSM kernel context
  - Fuzzers
- Moved to IGVM booting
  - Big step forward for generalizing HV-SVSM interface
  - Works on all platforms
  - Helpful for attestation
  - Supported on Hyper-V and Cloud Hypervisor
  - QEMU patches under review

SUSE

Progress

Platform Support

- Support for AMD SEV-SNP built-in
  - Lots of assumptions about SEV-SNP in the code-base
  - Work ongoing to generalize these parts
- Intel TDX Partitioning Support is evolving
  - Works different than AMD SEV-SNP
  - In-guest level switches
  - All #VEs cause exit to SVSM
- Native Platform
  - Support for running in non-confidential mode
  - Easier debugging
  - Make SVSM usable for VSM

SUSE

Progress

Services

- Support for vTPM emulation merged
- With vTPM SVSM became useful 🙂
- Ephemeral TPM
  - No persistence yet - EK is regenerated every boot
  - Trust established via SNP attestation report
  - Useful for runtime attestation
- Still a big chunk of C code running in kernel mode ☹️

SUSE

Progress

Road to User-Mode

- Many components implemented
  - Virtual memory management
  - RAM filesystem
  - Task concept and scheduler
  - Entry and exit code
  - ELF loader
  - Simple system call dispatcher
- Currently working on user-mode support code
  - Heap allocator
  - Basic file-system access library
- Still some way to go before usable
  - COCONUT as a Rust target would simplify things

SUSE

COCONUT-SVSM Plans

Plans

Finish User-Mode

- Implement user-mode support library
  - Heap allocator and basic file system access
  - Get started with the COCONUT Rust target
- System Call interface
  - Built around an everything-is-an-object model
  - Several object types like files, events, VMs, VCPUs, ...
- Create basic user-mode infrastructure
  - Simple init process
  - Move vTPM to user-mode

Plans

x2APIC Support

- One of the most urgent problems
  - Required for non-AMD platform support
- Main use-case is sending IPIs
  - TLB flushes
  - Remote function calls will simplify some of the SVSM logic
- Work has started, but there are features to implement in KVM first
  - Discussed later in the **Challenges** part
- IRQ support will use a TPR-based model

SUSE

Plans

Improve Kernel Isolation

- Finish virtual address space-split
  - Kernel: Global shared
  - Kernel: Per-CPU
  - Kernel: Per-Task
  - Uses: Per-Task
- Each kernel part gets its own heap
- This will nicely separate all execution contexts
- Use page-table self-map
- Get rid of the direct map

SUSE

Plans

Towards Paravisor Support

- The paravisor model enables non-enlightened OSes for Confidential Computing
- On TDX the paravisor model will likely be the default
    - TDX platform support work will get COCONUT closer to a paravisor model
- Paravisor model also planned for other architectures

SUSE

COCONUT-SVSM Challenges

Challenges

Overview

- Supporting AMD SEV-SNPs **Virtual Top-of-Memory (vTOM**) feature
- Fixing launch measurements: User/KVM **VMSA synchronization**
- **Privilege level support** in KVM (VMPLs/TDX Partitions/...)

SUSE

SUSE

Virtual Top-of-Memory (vTOM) for SEV-SNP

Challenges

Support for Virtual Top-of-Memory (vTOM)

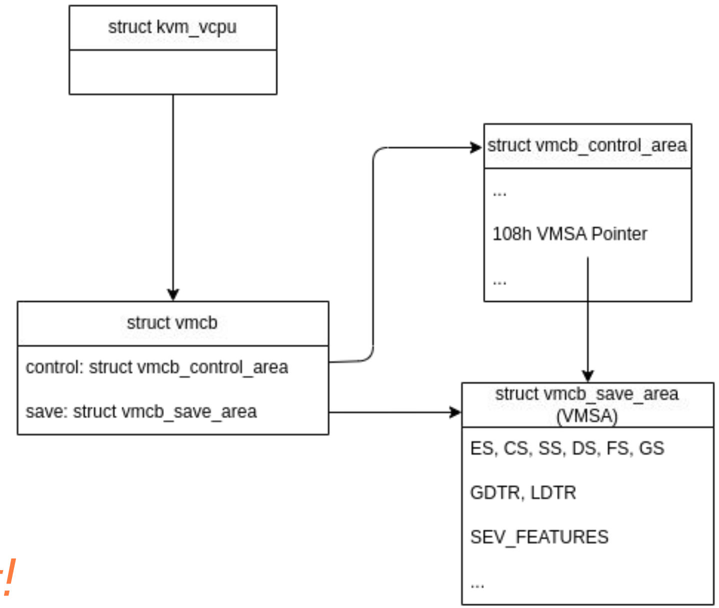
- Virtual Top-of-Memory is an AMD SEV-SNP feature
    - Introduces boundary in physical address space between always-encrypted/always-shared part
- Allows turning the page-table C(rypt)-bit into a S(hared)-bit
    - Alignment with TDX
- Requirement for paravisor support
    - Is helpful for plain SVSM setups too
- Needs support in host hypervisor
    - Have a way to mirror PGD entries in the nested page-table to both sides of the boundary

SUSE

Userspace/KVM SEV-ES VMSA Synchronisation

What is the VMSA?

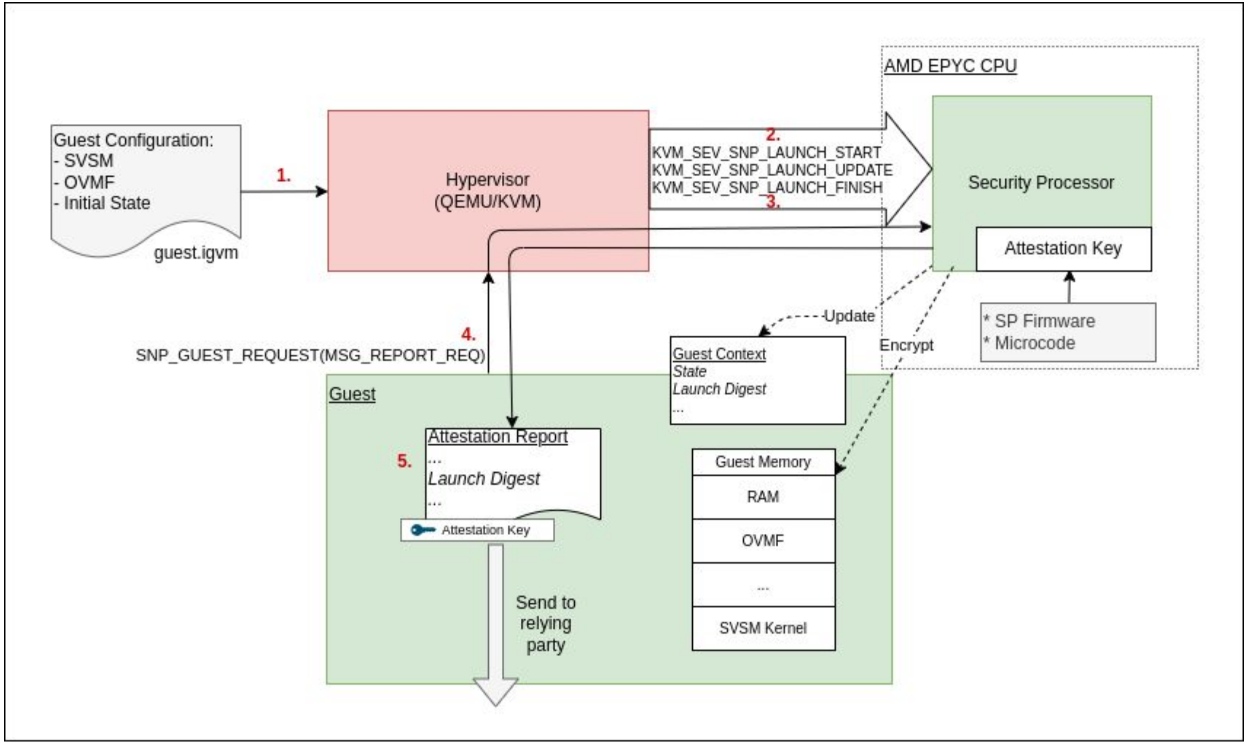
- X86 SVM guest configuration is defined using a Virtual Machine Control Block (struct vmcb)
- Consists of control area and save-state area
- Control area points to save state area
- KVM synchronises state between vcpu→arch and save area



CVMs must protect register state from host!

SEV-ES = Encrypted State

Launch Measurement

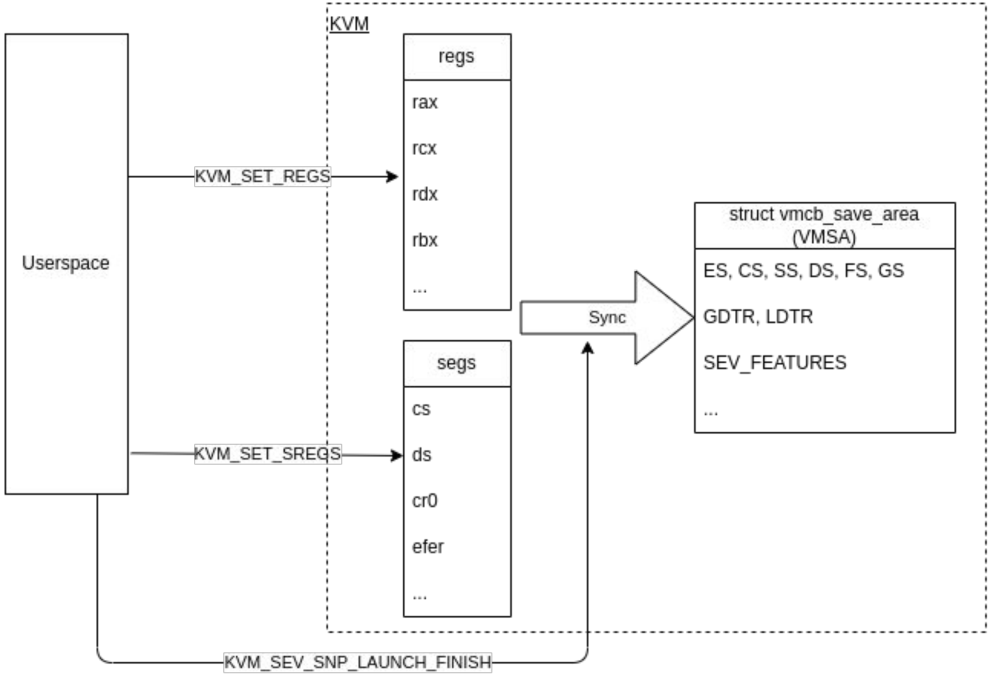


1. Guest owner provides configuration
2. Guest pages including VMSA are measured and encrypted
3. Launch measurement is finalized
4. Guest requests attestation report including launch measurement
5. Attestation report provides evidence

Changes to guest pages or VMSA will affect launch measurement

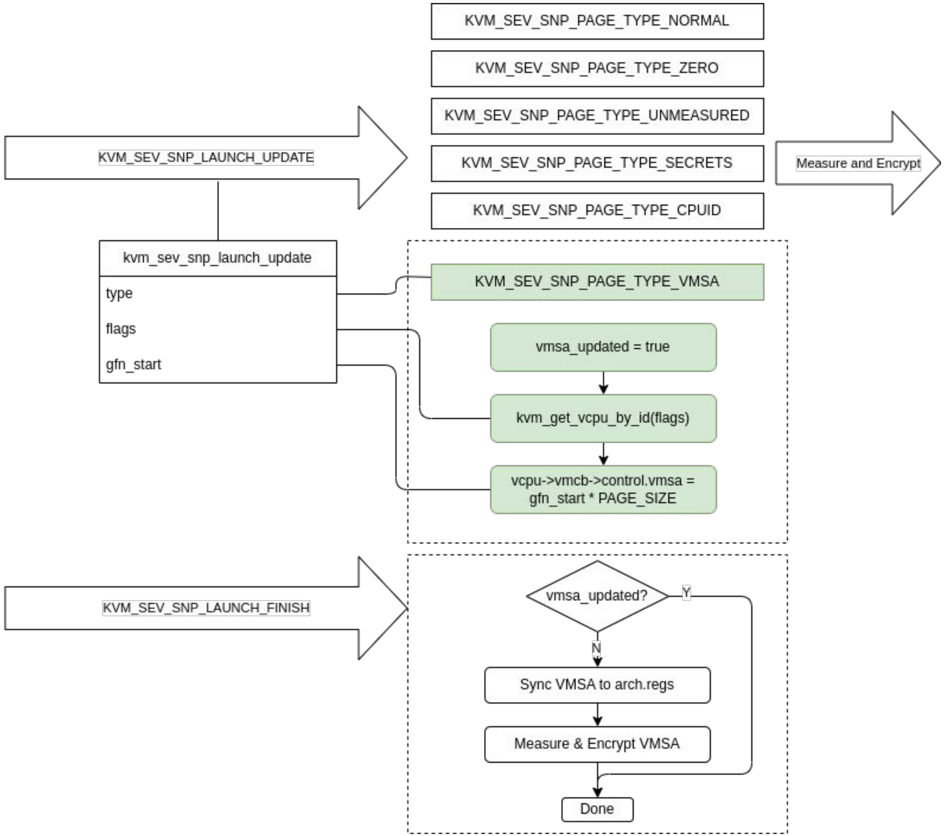
Setting the VMSA from Userspace

- No current way to directly set the VMSA
  - Registers must be set through KVM_SET_REGS and KVM_SET_SREGS
- KVM synchronises registers with VMSA on SEV launch finish
  - Problem: Not every field in VMSA is represented by KVM state
  - Also, GPA of VMSA is fixed in KVM
- Prediction of launch measurement is fragile



VMSA: A proposed solution

- Update existing KVM_SEV_SNP_LAUNCH_UPDATE IOCTL
  - Add support for VMSA page type
  - Set a flag to show VMSA has been provided by userspace
  - Need to provide vCPU ID via 'flags'
  - VMSA GPA is configured via 'gfn_start'
- Update KVM_SEV_SNP_LAUNCH_FINISH handler
  - If VMSA not provided via LAUNCH_UPDATE then sync and measure
  - Allows existing code to remain unchanged



What is left to complete?

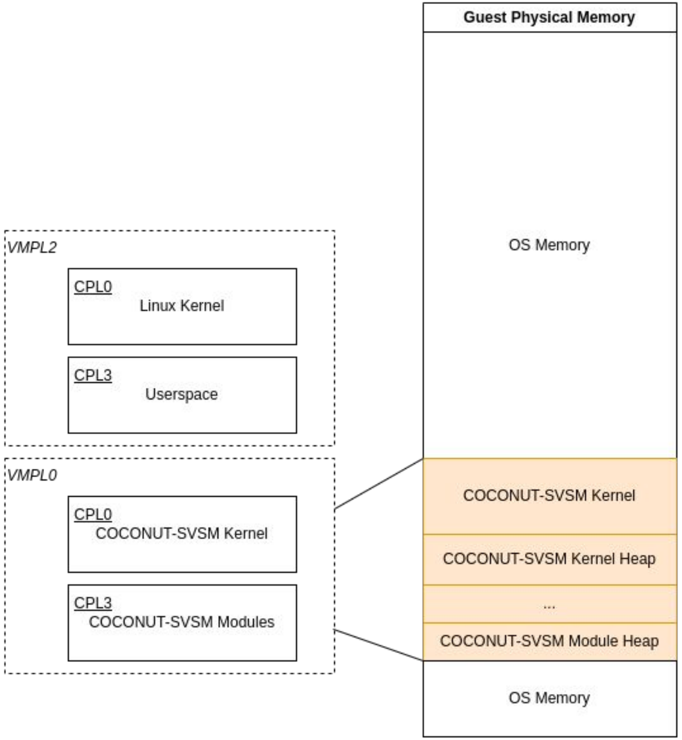
- Currently only support SEV-SNP is included in the patch
  - Need SEV-ES support
- VMSA state should be synchronised to KVM register state
  - Can affect the initial behaviour of the guest
  - No need to synchronise after the guest launches: state is encrypted

SUSE

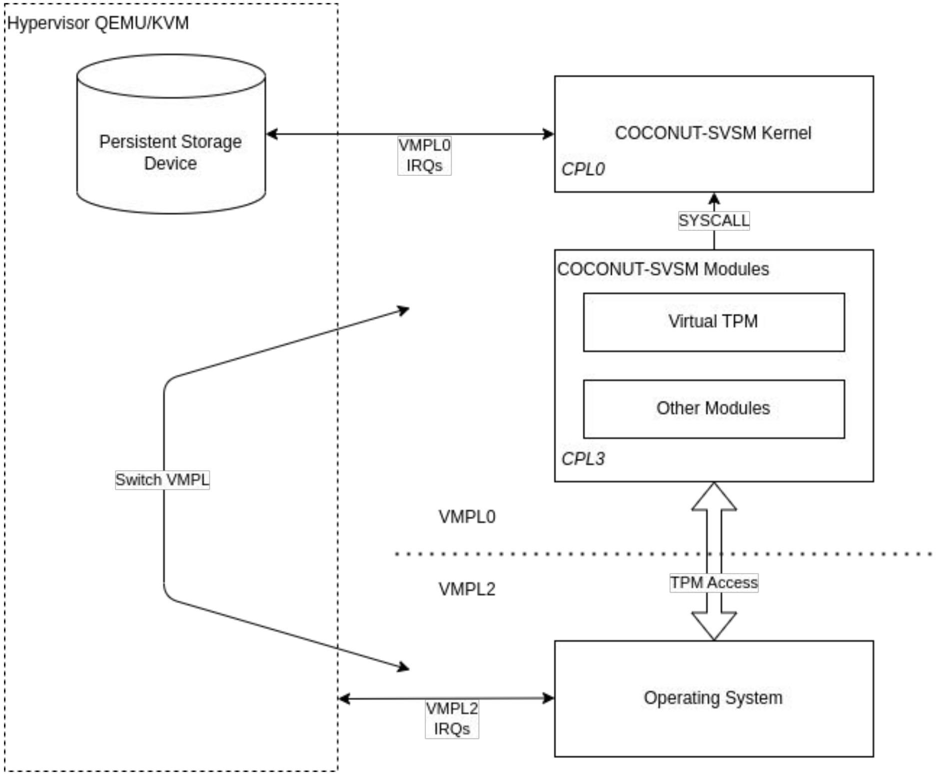
VMPLs in KVM

VMPL Overview

- VMPL = SEV-SNP privilege levels
  - Abstraction layers implemented in hardware
  - Similar to Virtual Secure Mode VTLs and TDX partitioning
  - SEV-SNP VMPL0 is the *highest* privilege level VTL0 is *lowest*!
- vCPUs are assigned a VMPL
  - VMPL0 has full access by default
  - Private guest memory access rights per VMPL
  - Lower VMPLs cannot access pages from higher VMPLs
- Allows VMs to create security boundaries
  - Restricting access to memory at VMPL > 0



Why do we need VMPLs in COCONUT-SVSM?



- SVSM: “Secure VM Service Module”
  - Allows modules (services) to be securely deployed in a VM
- Protection from host
  - Provided by memory and state encryption
  - SVSM integrity verifiable by launch measurement/remote attestation
- Protection from guest OS
  - Provided by running the OS at a lower-privilege VMPL (VMPL2)
  - SVSM runs at VMPL0
- Allows emulation of secure hardware, e.g. Virtual TPM
  - State and integrity of vTPM is protected

Without VMPLs, guest could manipulate state, compromising guest integrity.

SUSE

Implementing VMPLs in KVM

- No upstream support for VMPLs
- Much effort has already been undertaken
  - Much work done by Amazon
  - Implementations in both KVM and in Userspace
  - KVM: SEV-SNP support for running an SVSM - AMD
- Requirements:
  - A solution that supports multiple architectures: SEV-SNP, TDX, VSM and possibly more
  - High performance - VMPL switches can occur at a high frequency
  - TDX: VMPL switches can occur without a guest exit
  - Independent APICs for each VMPL: Restricted and Alternate injection

Implementing VMPLs in KVM

- AMD SEV-SNP SVSM patches already support VMPL switches
  - Simplest starting point for experimentation
  - Only limited state saved on VMPL switch - no APIC
  - Discussions at Linux Plumbers 2024 have found a potentially better way forward

<u>Backup/Restore VMPL state on each switch:</u>

- Create a per-VMPL save state area within ‘struct kvm_vcpu’
- On VMPL switch, backup vcpu fields into SSA for old VMPL, restore vpcu files from SSA for new VMPL
- Hard to maintain - every new addition requires code to backup/restore
- Hard to target non-current VMPL

<u>Create a per-VMPL ‘struct kvm_vcpu’:</u>

- Need to keep track of common state
- Associated via parent structure
- Share a single ‘struct kvm_run’
- On VMPL switch, simply select the ‘struct kvm_vcpu’ for the new VMPL from ‘struct kvm_vcpu_vmpl_state’.
- Easy to target non-current VMPL

Per-VMPL ‘struct kvm_vcpu’

- ‘Struct kvm_vcpu_vmpl_state’ tracks ‘struct kvm_vcpu’ for each VMPL
- Each VMPL `struct kvm_vcpu` for a vCPU points to the same `vcpu_parent`
- Fields that are common to all VMPLs are moved to `struct kvm_vcpu_common`
  - Only populated in VMPL0
  - All other VMPLs point to `&vcpu_parent->vcpu_vmpl[0]->_common`
- Unfortunately, all common field references need to be modified to be pointers
  - Many changes over many files

```
struct kvm_vcpu_vmpl_state {
        struct kvm_vcpu *vcpu_vmpl[4];
        int current_vmpl;
};

struct kvm_vcpu {
        struct kvm *kvm;
        struct kvm_vcpu_arch arch;
        struct kvm_vcpu_common {
                int cpu;
                int vcpu_id;
                int vcpu_idx;
                int mode;
                u64 requests;
                unsigned long guest_debug;
                struct mutex mutex;
                struct kvm_run *run;
                /* ... */
        } _common;
        struct kvm_vcpu_common *common;
        struct kvm_vcpu_vmpl_state *vcpu_parent;
        int vmpl;
};
```

SUSE

Implementing VMPL switches

KVM changes are fairly minimal when using the new structure layout:

- Creation/Destruction of a vCPU requires extra logic
  - Create/Destroy VMPL parent structure
  - Create and initialise or destroy struct kvm_vcpu for each VMPL
  - Setup links between structures
- LAPIC code requires target VMPL
  - Userspace does not specify a target VMPL for interrupts
  - Need to create a sensible default for now - VMPL2 for SEV-SNP
- SEV-SNP VMPL implementation based on Tom Lendacky's patch series
  - 'KVM: SEV-SNP support for running an SVSM'
  - Per VMPL fields replaced with individual fields in per-VMPL 'struct kvm_vcpu'
- Handling of VMPL switches from guest
  - Requests to change VMPL by guest are handled on exit from guest, updating current VMPL pointer.
  - Detection of change of VMPL while running guest occurs on exit from guest

What next?

- Look at Amazon’s PoC in detail
  - Add support for SEV-SNP
  - In-KVM optimisations - handling all switches in usermode will be far too slow

SUSE