The COCONUT Secure VM Service Module
An In-Guest Paravisor in Rust

KVM Forum 2023 - Jörg Rödel <jroedel@suse.de>
Why is an SVSM needed?

Confidential Computing Threat Model
Confidential Computing Threat Model

• In a confidential guest emulated parts of the hardware become untrusted
  • All emulated peripheral devices
    • Includes (X2)APIC, IOAPIC, network cards, disk controllers, TPMs, IRQ injection
  • OS needs hardening to not reveal secrets on malicious device input
  • Common pattern in confidential computing is to move HV functionality into guest context
Guest Device Emulation

• OS needs to be hardened against malicious device input

• Some devices carry security sensitive state (e.g. TPM)
  • Must be emulated in trusted guest context
  • Need memory isolation within the guest: VM Privilege Levels

• Additional software layer for in-guest emulation: SVSM
The COCONUT-SVSM
Some History

• Started in early 2022 - In parallel to linux-svsm

• Talked with AMD, but never reached the point where it made sense to switch over

• Linux-SVSM was announced August 29th, 2022

• COCONUT published on March 15th, 2023
In a Nutshell

- SVSM implementation in Rust
- Currently ca. 11500 LOC
- Focus on isolation within SVSM
- Uses support-code (Linux kernel and OVMF) from AMDs linux-svsm
- Currently running on AMD SEV-SNP
https://github.com/coconut-svsm/svsm
Features
SVSM Core Protocol

- Some operations of an AMD SEV-SNP guest are VMPL0-only
  - Page validation
  - Make memory available to OS VMPL
  - VCPU creation and deletion
- OS needs to call SVSM to perform these operations
Isolation Features

- SVSM was designed with strong focus on isolation
- Per-CPU page-tables
- WIP to enable execution of ELF binaries (modules) at CPL-3
  - Modules provide additional services to OS
  - Device emulation in modules (e.g. TPM)
- IPC mechanism TBD
SVSM Modules

• Running modules at CPL3 allows separation between core SVSM and third-party code

• A vTPM can be entirely written in C and not interfere with the Rust code base of the SVSM
Modules at CPL-3 WIP

- Supporting user-space execution environment needs some boilerplate code
  - Task management and scheduling (cooperative vs. preemptive)
  - Task switching
  - Task memory management
  - Syscall interface

- Progress can be tracked at https://github.com/coconut-svsm/svsm/issues/16
Module Use-Cases

• With a TPM we have secure runtime attestation in a confidential guest

• Other possible modules
  • UEFI variable store
  • Attestation
  • Other device emulations
  • Core protocol in a module?
More Features

- Boots in 32-bit protected mode using a two-stage loader
- Global memory allocator using buddy and slab algorithms
- ELF loader
- Console support with switchable backends
Debug Features

- Serial console support
- Collect and print backtraces
  - Prints a backtrace on panic
- Can collect stack-traces without printing them - useful for lock debugging
- Optional GDB stub
Future Directions
SVSM Boot Process

- Presented to guest as additional option ROM after BIOS and VARS
- Launched in 32-bit protected mode

640k ISA RAM up to 3 GiB PCI SVSM OV MF RAM
SVSM Boot Process

- Current boot process not optimal
  - Uses a hard-coded non-standard initial CPU state
- Option 1: Bundle FW into SVSM and launch it from ROM index 0 using default reset vector
- Option 2: Load initial VM state from a single file (memory and register state, VM parameters, ACPI tables, memory map)
  - Works as a cross-hypervisor interface
- Final decision needs to take other architectures into account
Persistency Layer

- Secure storage space for SVSM modules
- TPM state
- UEFI variables
- Encrypted and integrity protected
- HV↔SVSM interface: Block vs. file/object based?
Interrupt Proxy?

- Support for restricted injection unlikely to land in Linux soon
- Let the SVSM take IRQs via Restricted Injection
- Forward IRQs from SVSM to OS via Alternate Injection
- Requires (at least partial) APIC emulation in SVSM (for managing TPR/EOI/IRR updates)
Validation Bitmap?

- Let SVSM keep track of guest accepted/unaccepted memory
- Move checks into SVSM too
- Makes it simple to preserve bitmap across OS reboots/kexec+kdump
Towards Unenlightened OSes

- Add ReflectVC support, turning SVSM into a paravisor
- Running device emulations as modules
- vTOM support
- More complex SYSCALL interface needed
Thank you!

Questions?
SEV-SNP VM Privilege Levels

- Hardware feature in AMD SEV-SNP capable processors
- 4 levels, VMPL0-VMPL3
- Allow memory isolation within confidential guest VMs
  - Read/Write/Execute permissions per VMPL level
  - Can be used to hide memory from the operating system
SEV-SNP VM Privilege Levels

CPL0  CPL1  VMPL3  CPL2  CPL3
CPL0  CPL1  VMPL2  CPL2  CPL3
CPL0  CPL1  VMPL1  CPL2  CPL3
CPL0  CPL1  VMPL0  CPL2  CPL3
Moving OS to Higher VMPL

VMPL0

VMPL1

VMPL2

VMPL3

SVSM