[DRAFT] COCONUT-SVSM Development Plan

Contents

1 First Principles of COCONUT-SVSM Design

The first section contains the overall principles applied when creating the design of COCONUT-SVSM. The following sections break down known development items and their dependencies to achieve these principles.

1.1 Mission

The mission of COCONUT-SVSM is **be a platform to provide secure services to Confidential Virtual Machines (CVMs)**.

The services provided by COCONUT-SVSM aim to increase the security of the CVM by:

- Moving hypervisor services from the untrusted host into the trusted CVM context.
- Handle CVM specifics in the SVSM instead of the requiring additional support in the OS to reduce the attack surface of the guest operating system.

1.2 Security

COCONUT-SVSM is one of the most critical parts in the security architecture of a CVM. Therefore any design decisions have to take security implications into account.

The main tool used to achieve better security properties within its own code-base is *Isolation*. In particular, this means:

- Services provided to the CVM run as **user-mode processes** by default. Only when there are very good reasons parts or whole services can be implemented in the COCONUT kernel.
- Memory isolation within the COCONUT kernel. Provide per-CPU and per-Task memory which is not accessible outside of its context.
- Cryptographic isolation: Isolate the cryptographic code and data (keys) from the rest of the system and between contexts.

1.3 Execution Modes

The COCONUT-SVSM platform aims to support three execution modes:

- **Enlightened OS mode**: In this mode the guest OS is aware of the environment and can handle most CVM specifics itself. The guest OS has a VE/VC exception handler and manages private and shared memory. The SVSM provides services to the guest OS which can not be securely provided by the hypervisor, like emulating devices with security-sensitive state.
- **Paravisor mode**: This mode is for running guest operating systems which have limited or no support for handling CVM specifics. The SVSM is responsible for handling VE/VC exceptions on behalf of the guest OS and manage private and shared memory. In addition to that the SVSM will still emulate security sensitive devices on behalf of the host hypervisor.
- **Service-VM mode**: When running in this mode, the SVSM is the operating system of the CVM and does not run alongside another guest

OS in the same TEE. The services are provided to other CVMs via a hypervisor-provided communication channel.

1.4 Multiple Platform Support

Support for multiple platforms is another major goal of COCONUT-SVSM. Platforms include multiple hardware platforms like AMD SEV-SNP, Intel TDX and ARM CCA as well as multiple hypervisor platforms like QEMU/KVM and Hyper-V.

The following sections list the planned or in-progress development items needed for COCONUT-SVSM to achieve its mission and principles.

1.5 Rust is the Default Programming Language

Unless otherwise noted the whole COCONUT-SVSM code base is written in the Rust programming language. This includes the COCONUT kernel and all user-space libraries and binaries.

2 [BldSys] Build System

The COCONUT-SVSM platform needs a powerful build system which is capable of building the kernel and user-mode components as specified by a build recipe. The recipe is provided in a editable and machine readable format, e.g JSON or YAML. The build system will package user-mode components into the RAM file-system image and bundle it with the kernel and firmware into the output IGVM file.

As an enhancement (not needed in the first step) the build system also needs the capability to build or include user-mode components which are not part of the COCONUT-SVSM repository.

3 [UsrMode] User-Mode Support

This section lists the work items to implement support for running services in user-mode.

3.1 [HeapAlloc] Heap Allocator

User-mode binaries need dynamic memory allocation. This will be provided by a heap allocator which supports all necessary allocation sizes and can be used from non-rust user-mode code as well. This means that the size of the allocation is not required as an input to a free operation.

3.2 [SyscallAbi] Define a System Call ABI

Make definitions for how system call parameters are communicated between user-space and the COCONUT kernel. Design all data structures for user-kernel communication in a way that is usable with other programming languages as well.

3.3 [UserLib] User-mode Support Library

Depends on: **UsrMode.HeapAlloc**, **UsrMode.SyscallAbi**

The user-mode support library provides support to develop and build user-mode binaries for COCONUT-SVSM. The library contains:

- Linker script.
- Platform setup code
- Heap allocator
- Syscall APIs
- All future SVSM specific user-mode interfaces

While the library will be written in Rust, it should support to be used from other programming languages like C and C++. This has implications for the public data structures and function names.

3.4 [SysCallBatch] Define SYSCALL batching Mechanism

Depends on: **UsrMode.SyscallAbi**

In a paravisor setup it will become necessary to handle a larger number of system calls to fulfill requests. Issuing single system calls can become a performance problem, so a batching mechanism to allow sending multiple system calls within one request is needed.

3.5 [FsSysCall] File-System System Calls

Depends on: **UsrMode.SyscallAbi**

Implement user-mode APIs to interact with the filesystem. This includes opening, reading, writing, memory-mapping, and closing files.

3.6 [IpcEvt] IPC and Event Delivery Framework

Depends on: **UsrMode.SyscallAbi**

User-mode and kernel components in COCONUT-SVSM need communication interfaces to send and receive data and events. A framework enabling the communication needs to be designed and implemented in the COCONUT kernel.

3.7 [InitTask] Create Init Task

Depends on: **UsrMode.UserLib**, **UsrMode.FsSysCall**

Implement an user-mode process for the SVSM which is launched as the first user-mode process by the COCONUT kernel. It is responsible for setting up the execution environment and launches other user-mode services as specified by a configuration file provided with the RAM file-system.

3.8 [ReqLoopUser] Move Request-Loop to User-Mode

Depends on: **UsrMode.InitTask**, **UsrMode.IpcEvt**

Create a simple user-mode process which executes the request-loop for SVSM protocol requests in user-mode. Initially most of the actual handling can stay in kernel-mode, but this process is a starting point to move most of request parsing and handling to user-mode as well.

3.9 [IdleTask] Let the Idle Task idle

Depends on: **UsrMode.ReqLoopUser**, **UsrMode.IpcEvt**

Currently the idle tasks in COCONUT-SVSM are the ones switching to the guest OS in a less privileged level. Move that code into a system call and execute it in the request-loop service. The idle tasks should then just idle and halt the execution.

3.10 [Fpu] Support FPU Instructions

Implement support to save and restore FPU state for user-mode processes and switch FPU state at task-switch. Also implement an API to enable FPU usage in the COCONUT kernel.

3.11 [VTpmUser] Move vTPM to User-Mode

Depends on: **UsrMode.InitTask**, **UsrMode.IpcEvt**

Move the vTPM emulation code into a user-mode service.

3.12 [VmmIf] Define VMM Interface

The COCONUT kernel needs to provide a VMM-like interface for user-mode processes to control the execution of the guest operating system. This interface should be flexible enough to support the **Enlightened OS Mode** and **Paravisor Mode** of operation.

This interface will also allow to run deployment specific versions of VM management tasks in user-mode.

4 [IrqSec] IRQ Security

COCONUT-SVSM can help to provide secure IRQs to guest operating systems.

4.1 [HvIrqSig] Specify Platform-Agnostic Hypervisor to SVSM IRQ Signaling Interface

The SVSM targets an architecture where each CVM privilege level is provided an independent IRQ vector space. The host hypervisor or the hardware does not or can not always emulate a separate LAPIC for each privilege level, which means a defined communication standard between the hypervisor and the CVM is needed. The standard needs to define data structures and algorithms for the hypervisor to report IRQ events for individual CVM privilege levels.

4.2 [Apic4Svsm] X2APIC Support for COCONUT-SVSM

On the x86 architecture, the SVSM runs in the highest privilege level of the CVM and has its own IRQ vector space, usually provided via a hardware- or hypervisor-provided APIC. Support code for the APIC is required so that the SVSM can send IPIs between VCPUs.

4.3 [ApicEmul] vXAPIC and vX2APIC Emulation Support

Depends on: **IrqSec.HvIrqSig**, **IrqSec.Apic4Svsm**, **UsrMode.IpcEvt**

Add an emulation for vXAPIC and vX2APIC to the SVSM for use by the guest OS.

4.4 [IrqDispSvsm] COCONUT-SVSM IRQ Dispatch Code

Depends on: **IrqSec.Apic4Svsm**

COCONUT-SVSM needs infrastructure to dispatch injected IRQ events to itself. This includes the ability to register IRQ handlers for specific vectors. For the AMD platform IRQ delivery in the presence of *Restricted Injection* is also needed.

4.5 [IrqDispGuest] Guest IRQ Dispatch Code

Depends on: **IrqSec.HvIrqSig**, **SevSnp.SnpAltInj**

For setups where COCONUT-SVSM forwards injected IRQs to guest operating systems it needs support to determine when a guest OS is ready to accept the IRQ and inject it into the corresponding privilege level.

5 [SevSnp] Support for AMD SEV-SNP with VMPLs

The AMD SEV-SNP hardware extension is the bring-up platform for COCONUT-SVSM and support is not yet finished.

5.1 [EnlightOS] Finish Enlightened OS Mode

Depends on: **UsrMode.VTpmUser**, **UsrMode.VmmIf**

Finish support to run COCONUT-SVSM as a service platform for enlightened guest operating systems. This mostly relies on moving the existing services to user-mode.

5.2 [SnpAltInj] Alternate Injection Support

Depends on: **IrqSec**

Support taking notifications for IRQs to lower privilege levels in the COCONUT kernel and use the *Alternate Injection* feature to inject the IRQs into the guest OS.

5.3 [SnpParaV] Support Paravisor Mode

Depends on: **ParaV**, **UsrMode.VmmIf**

Enable support for the *ReflectVC* feature of AMD SEV-SNP to allow offloading VC exception handling from the guest OS into COCONUT-SVSM an run (mostly) un-enlightened operating systems.

6 [TdxPart] Support for Intel TDX-Partioning

The Intel TDX with Partitioning support is the second major platform COCONUT-SVSM aims to support.

6.1 [Igvm] IGVM Boot

The first step to support the TDX platform in COCONUT-SVSM is to implement boot support via an IGVM platform file. This needs support in the COCONUT kernel as well as in the QEMU IGVM loader.

6.2 [PlatAbstr] Platform Abstractions

The COCONUT kernel contains a lot of hard-coded SEV-SNP assumptions. These need to be abstracted into a generic API which can be implemented for multiple platforms.

6.3 [TdxSmp] Multi-processor Support

Booting multiple CPUs in a TD guest needs some modifications in the COCONUT kernel as on Intel the TD vCPUs start from a fixed address.

6.4 [Boot] TDX Boot support

Depends on: **TdxPart.Igvm**, **TdxPart.PlatAbstr**, **IrqSec.Apic4Svsm**, **TdxPart.TdxSmp**

Implement a platform API backend to boot COCONUT-SVSM in an Intel TD with partitioning support.

6.5 [TdxParaV] TDX Paravisor Support

Depends on: **ParaV**, **UsrMode.VmmIf**

Implement support for running un-enlightened guest operating systems in an Intel TD using TDX partitioning. It is fine to implement this alongside generic paravisor support.

7 [ParaV] Paravisor Support

Besides enlightened guest operating systems COCONUT-SVSM should support un-enlightened operating systems as well. This requires a lot of new functionality to offload CVM specific handling from the OS into the SVSM.

7.1 [SysCall] SYSCALL Interfaces

Depends on: **UsrMode.SysCallBatch**, **UsrMode.UserLib**

Part of: **UsrMode.VmmIf**

Define and implement system call interfaces which user-mode code can use to read and modify state of the guest OS. This includes CPU, memory, and IRQ states.

7.2 [InstrDec] X86 Instruction Decoder

The SVSM needs an instruction decoder to handle events from the guest OS which were triggered by specific instructions. An incomplete lists of events:

- CPUID
- RD/WRMSR
- MMIO
- IOIO

For MMIO only a minimal subset of instructions is supposed to be supported to keep the instruction decoder and its attack surface small.

Later, and if more instructions need to be supported, a user-mode extension to the SVSM in-kernel instruction decoder can be discussed.

7.3 [MmioDisp] MMIO/IOIO Event Dispatch Framework

Depends on: **ParaV.InstrDec**, **UsrMode.IpcEvt**

Part of: **UsrMode.VmmIf**

A framework is needed to dispatch MMIO and IOIO events to different user-mode services or kernel-mode components, based on the MMIO address or IOIO port-range targeted by the access.

7.4 [EvtHndlr] User/Kernel VE/VC Event Handlers

Depends on: **UsrMode**, **UsrMode.SysCallBatch**, **ParaV.MmioDisp**

Implement handlers in user- or kernel-mode for all possible VE/VC events triggered by the guest OS. The default target is user-mode, only handling events in kernel mode when there are very good reasons for it (e.g. performance).

8 [Crypto] Cryptography Support

The development items in this section are no pre-requisite to items in other sections which need cryptography. Initially other parts of COCONUT-SVSM can use their own cryptography libraries and be converted to a common implementation once it is ready to use.

There are two major design goals for the cryptography layer:

- *Isolation*: Make sure the cryptographic keys are not accessible outside of the crypto layer. The keys of different users also need to be isolated from each other.
- *FIPS certification*: The crypto layer and library needs to be designed in a way the can be certified by FIPS. This means the library needs to be a standalone binary which can be executed separately for any given context in need for cryptography.

8.1 [Lib] Provide Crypto Library

Implement or port a crypto library to the COCONUT-SVSM platform. FIPS certifiability is not initially required, but development should be targeted towards this goal.

Ideally a crypto library is written in Rust, but that is not a strict requirement.

8.2 [Context] Design Isolation Context for Cryptographic Code

Depends on: **Crypto.Lib**

Design and implement an execution context and interfaces to interact with it. The implementation needs to provide the isolation capabilities listed above.

9 [Persist] Persistence

One of the main use-cases for the SVSM is to emulate devices containing security sensitive state in a trusted environment. In order for the security sensitive state to be persistent across restarts of the CVM instance, a persistency layer is needed.

9.1 [EarlyAttest] Define and Implement Early Attestation Architecture

A process for early attestation and key delivery, based on hardware attestation capabilities, is needed. The attestation is used as a proof to a Key Broker Service (KBS, the relying party) that the CVM is in an expected state. Based on the attestation result the KBS will provide secrets (like a key for persistent storage) to the SVSM.

9.2 [BlkLyr] Block Layer

The COCONUT-SVSM will need to support different storage backends for persistent storage. In order to have a common interface to all supported hypervisors, a generic block layer is needed which is the front-end to specific backend implementations. Encryption and integrity protection of the storage will also be implemented on the block layer.

9.3 [FS] File System for Persistent Data

Depends on: **Persist.EarlyAttest**, **Persist.BlkLyr**

A simple file-system driver is needed to support persistence for multiple services and device emulations. Design is TBD, but there is likely no need to support directories.

9.4 [FsPerm] Permission Model for File System Data

Design and implement a permission model for data on the file system which allows to limit which persistent data is accessible by a given user-mode process.

10 [StdRust] COCONUT-SVSM as Rust Tier3/2 Target

Depends on: **UsrMode.UserLib**, esp. **UsrMode.HeapAlloc**

To make it easier to develop new user-mode modules and bring the Rust standard library to the COCONUT-SVSM ecosystem, support for a COCONUT platform target in the upstream Rust project is desired.

10.1 [Tier3] Initial Support as a Rust Tier-3 Target

Depends on: **UsrMode.HeapAlloc**, **UsrMode.FsSysCall**, **UsrMode.IpcEvt**

This will be the initial step of supporting COCONUT-SVSM as a platform target in Rust. Minimal support is needed from the SVSM side, like a heap allocator and basic file-system APIs. The interfaces to the SVSM kernel do not need to be stabilized yet.

10.2 [Tier2] Towards Tier-2 Target Support

Depends on: **StdRust.Tier3**

Stabilize the SVSM kernel interfaces and organize code in a way to reach status as a Rust Tier-2 target platform.

11 [Sec] Securing COCONUT-SVSM Code Base

This section lists a loosely coupled list of work items to improve the security of the COCONUT-SVSM platform.

11.1 [UsrSec] User-mode Security Framework

Define and implement a security framework which allows to limit the capabilities of user-mode processes to interact with the SVSM kernel. In Linux terms this would be similar to SELinux.

11.2 [ValidState] Track Validation State per 4KiB Page

In order to mitigate a various possible double-validation attacks for memory pages, the COCONUT kernel needs to track the validation state of each 4KiB page in the system. Implement the data structures and integrate the checks in the page validation backends.

11.3 [CodePatterns] Fixing Unsound Code Patterns

The GitHub issues for the COCONUT-SVSM contains an issue which lists unsound code patterns. This list needs to be updated, evaluated and the patterns need to be fixed.

11.4 [PtSelfMap] Page Table Self Map

Implement a self-mapping of the page-table to simplify modifications. This allows to implement a fast `virt_to_phys()` mapping without a direct-map.

11.5 [NoDirectMap] Getting Rid of Kernel Direct-Map

Depends on: **Sec.PtSelfMap**, **UsrMode.HeapAlloc**

The COCONUT kernel currently uses a direct map of VMPL0 physical memory. The direct map is contrary to the isolation goals of COCONUT-SVSM and should be removed in order to increase security of the overall architecture and achieve actual isolation.

This is a multi-step approach which requires a rewrite of the page allocator and the way heap allocation works. Allocation and usage of shared memory will also fundamentally change.

11.6 [Fuzzing] Improve Fuzzing

The COCONUT-SVSM repository contains a good number of fuzzers already for parts of the code-base. Build on that and extended the fuzzers over time to cover more or most code of the COCONUT-SVSM platform.

11.7 [Stress] Adding Stress-Tests

This is related to fuzzing, but targeted at a fully running COCONUT-SVSM instead of individual parts of the code. Stress tests need to be implemented to find any kind of issues in the kernel and user-mode code, especially race conditions, lock inversions, and so on.