Consideration about enabling hypervisor in open-source firmware

OSFC 2019

Piotr Król

3MDEB

About me

Piotr Król
Founder & Embedded Systems Consultant

- open-source firmware
- platform security
- trusted computing

@pietrushnic
piotr.krol@3mdeb.com
linkedin.com/in/krolpiotr
facebook.com/piotr.krol.756859

Agenda

- Introduction
- Terminology
- Hypervisors
- Bareflank
- Hypervisor as coreboot payload
- Demo
- Issues and further work

Introduction

Goal

create firmware that can start multiple application in isolated virtual environments directly from SPI flash

Motivation

- to improve virtualization and hypervisor-fu
- to understand hardware capabilities and limitation in area of virtualization
- to satisfy market demand

Terminology

Virtualization is the application of the layering principle through enforced modularity, whereby the exposed virtual resource is identical to the underlying physical resource being virtualized.

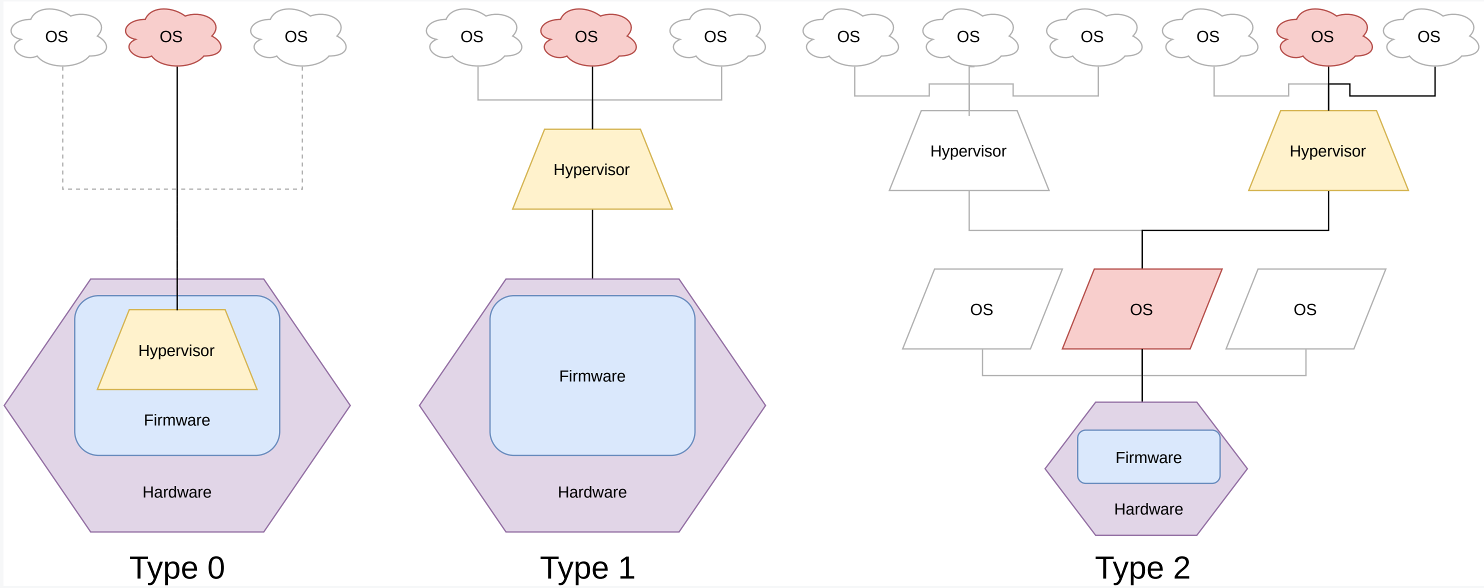
- layering - single abstraction with well-defined namespace
- enforced modularity - guarantee that client of the layer cannot bypass it and access abstracted resources
- examples: virtual memory, RAID

VM is an abstraction of complete compute environment

Hypervisor is a software that manages VMs

VMM is a portion of hypervisor that handle CPU and memory virtualization

Edouard Bugnion, Jason Nieh, Dan Tsafrir, Synthesis Lectures on Computer Architecture, Hard- ware and Software Support for Virtualization, 2017



- Type 2: VMware Player, Oracle VirtualBox, QEMU
- Type 1.5(?): Linux KVM, FreeBSD bhyve
- Type 1: Xen, Microsoft Hyper-V, VMware ESX/ESXi, Bareflank
- Type 0: "corevisor", IBM LPARs(Logical PARtitions) and Oracle LDOMs(Logical Domains), L4Re

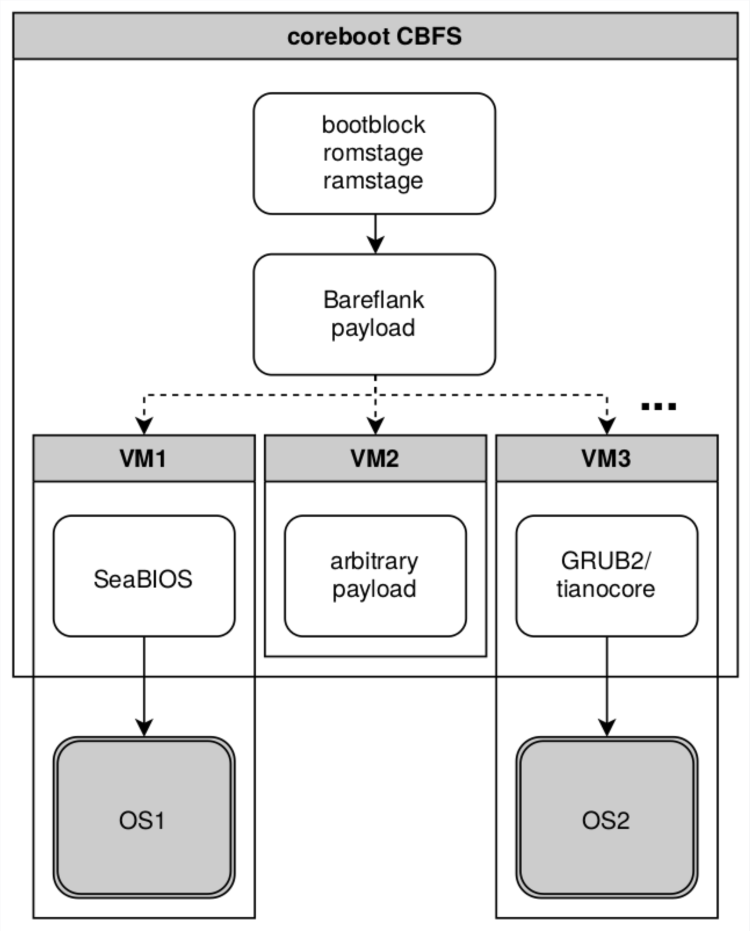
Embedded bare-metal hypervisor features

- relatively new to industry (early 2000s)
- efficiency
  - typically small and fast
  - minimal impact on system
- security
  - helps to minimize TCB (*Trusted Computing Base*)
  - easier certification and reliability testing
  - subsystem encapsulation
- communication
  - implementation specific
  - typically: VMCALL, VMFUNC, CPUID, MSR, EPT manipulation for bigger piece of data
- isolation and real-time capabilities
  - minimal delay caused by software
  - performance close to hardware native capabilities

Embedded hypervisor applications

- Mission and safety critical applications (Robotics/Automotive/Medical/Military)
  - strong isolation of non-critical and critical computation
  - isolation of fault detection components
- Legacy code re-use
  - one VM for legacy code, other for new features
  - migration from uni-core to multi-core systems
- Manageability
  - hypervisor may expose additional management layer even with remote access
  - dynamic system updates

Hypervisor in a firmware



Bareflank hypervisor

- lightweight hypervisor SDK written in C++ with support for Windows, Linux, UEFI and **coreboot**
- Lead by Assured Information Security, Inc.
- Supports Intel, but ARM and AMD are planned in future releases
- Most important features:
  - Support logic (memory manager, serial, libc++)
  - Virtual CPU Management
  - Virtualization Extension Logic
- After scaffolding new hypervisor most work is related to implement or modify handlers

Bareflank getting started

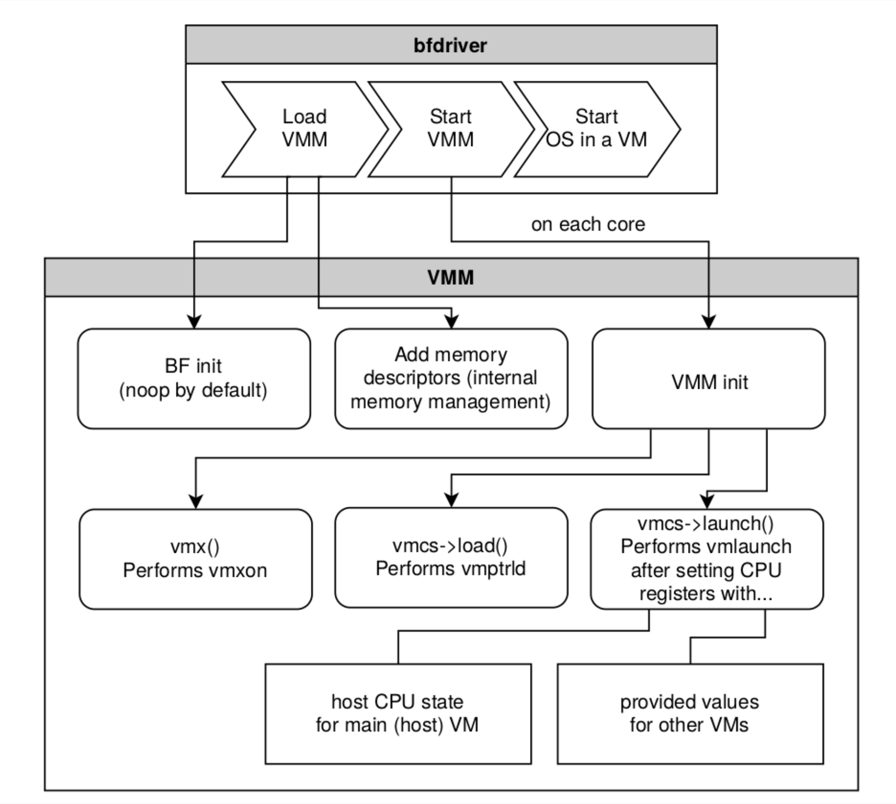
- Project is quite quickly moving target
    - massive code changes can happen
    - it works without any further improvements so it is wise to stick to one particular version during development
    - hypervisor in a firmware update needs reasonable justification
- Instruction to build and test your first VM are straight forward
- Community is very supportive with short response time
- Building
    - modern C++ dependencies can be challenging
    - https://github.com/3mdeb/bareflank-docker
    - instructions for UEFI, but VMM can be used with coreboot
    - build process produces `vmm.h`

Bareflank payload

- VMM - code delivered as a C header (`vmm.h`) file with bytecode as a result of Bareflank SDK build
  - 88k SLOC of bytecode (~1.2MiB), without any customization
- bfdriver (Bareflank driver) - minimal C code providing necessary hypervisor hooks and code for VMM launching delivered in 3mdeb coreboot fork as payload
  - 5.7k SLOC
  - ~1k SLOC of libpayload modifications
  - ~800 SLOC really written, rest are headers and common code from Bareflank

```
user:coreboot git:(bareflank_payload) $ tree payloads/bareflank -L 1
payloads/bareflank
├── common.c    // directly from Bareflank
├── entry.c     // code to start VMM and payload
├── include     // bunch of includes directly from Bareflank project
├── Kconfig
├── Makefile
└── platform.c  // platform specific code for memory handling
```

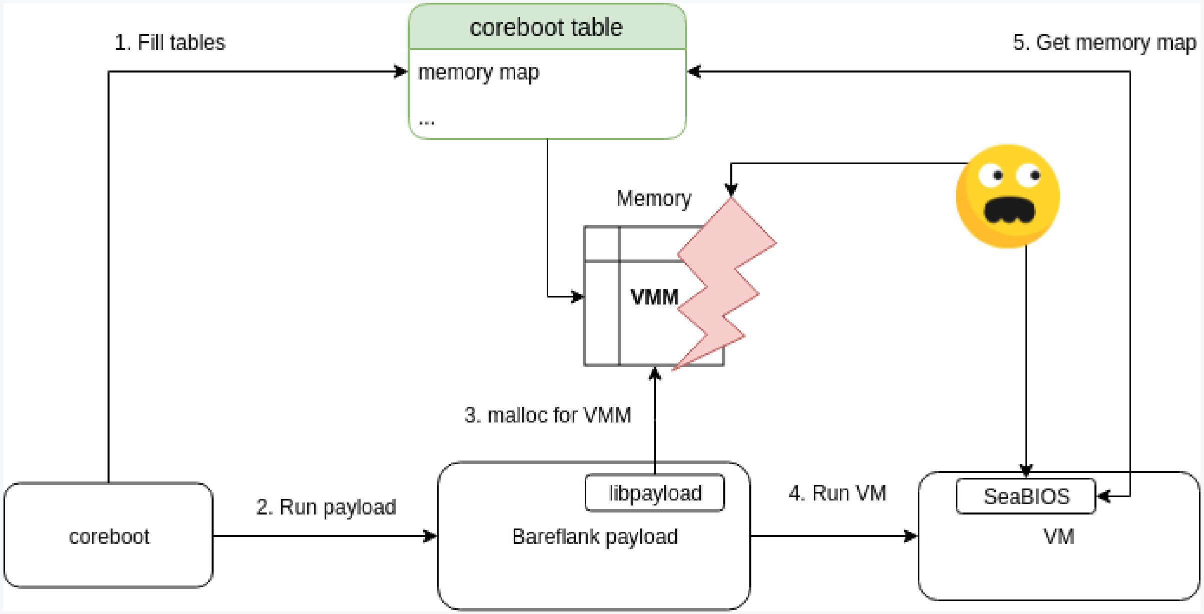
Bareflank initialization flow



Hypervisor as coreboot payload

- By default Bareflank uses
  - OS specific API if used as type-2 hypervisor
  - UEFI Boot Services if used as type-1 hypervisor
- Libpayload for the rescue
  - x86_64 support added
  - x86_64 exception handling
  - x86_64 drivers may still need some fixes
- Size problems
  - we need something that will start in VM (our choice was SeaBIOS)
  - continuous SPI space was required (310kB after LZMA compression, 3.5MB uncompressed)

coreboot tables

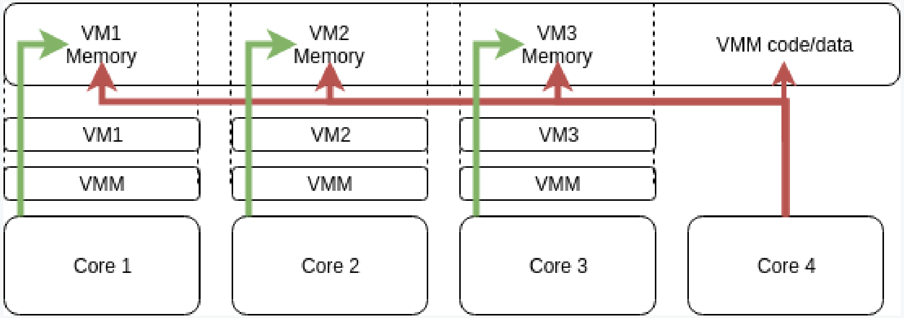


- SeaBIOS use coreboot tables provided memory map
- libpayload doesn't modify memory map in coreboot tables
- this result with SeaBIOS being able to overwrite VMM memory
- coreboot was extended to reserve memory at compile-time
    - it also helps in limiting memory available to VMs

32 and 64-bit game

- Bareflank requires 64-bit mode
- coreboot works in 32-bit mode for now
- SeaBIOS which we use as payload is in 32-bit mode
- switching back and forth can be tricky, but we managed to implement that flow
- Meanwhile we found some problems in libpayload:
    - incorrect assumptions that `sizeof(size_t) == sizeof(uint32_t)` in CBFS handling code: `payloads/libpayload/arch/x86/rom_media.c`
    - we changed order of entries in GDT since it didn't match coreboot structure - looks like problem synchronizing coreboot and libpayload

VMX for each core

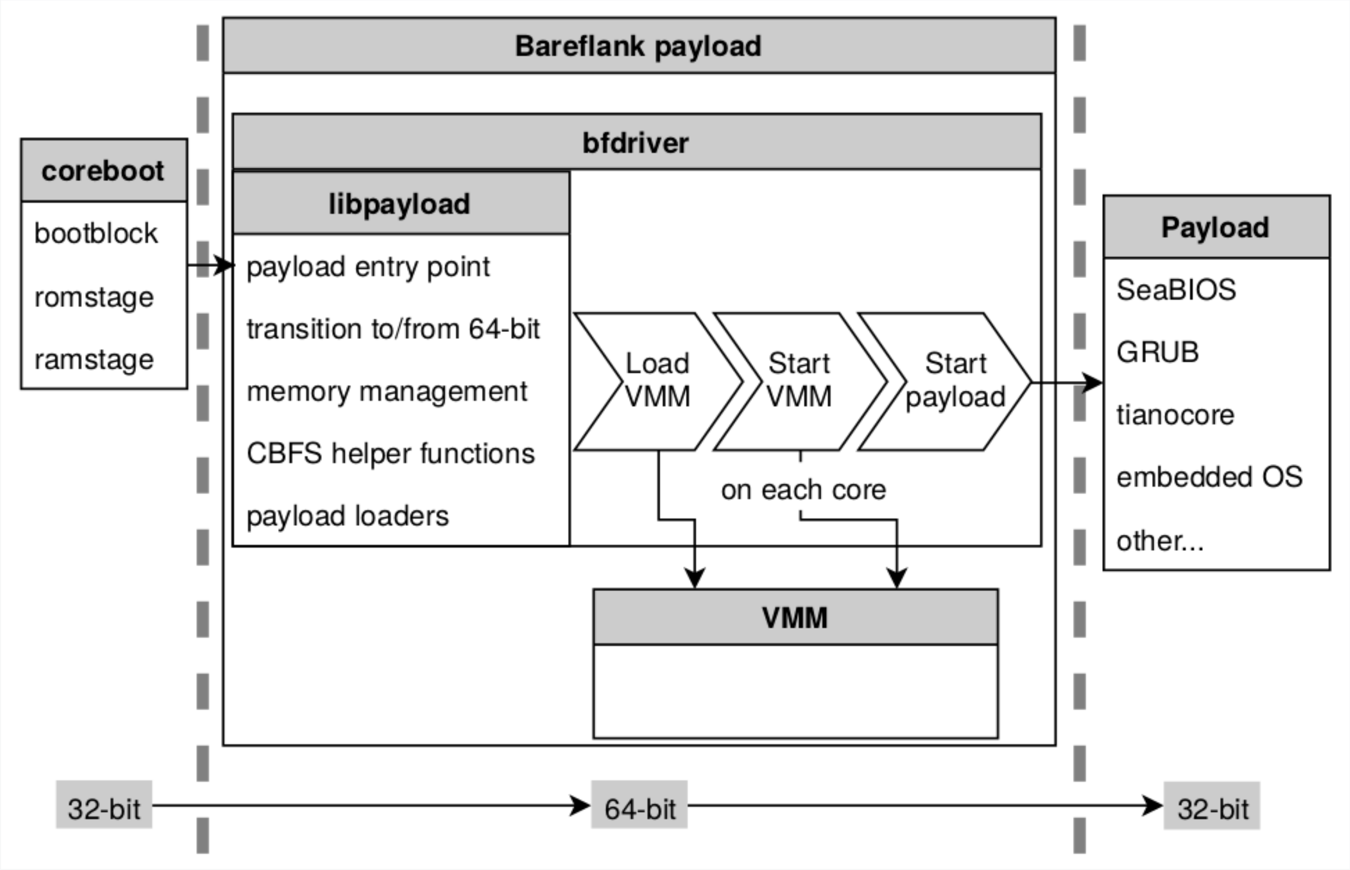


- each core have to setup VMX mode separately
  - this give flexibility in VM exit events handling
  - one VM may require more physical hardware access than other
- not running VMX on all cores may cause security risk
  - code running on core without VMX is beyond control
  - core that has no VMX running can access VMM memory and communicate with all devices
- Libpayload to the rescue again
  - we implemented MP code that gives ability to execute on given core/thread

Memory reservation and EPT

- based on information in coreboot tables SeaBIOS will create memory tables marking hypervisor code as reserved
- thanks to that VM knows which parts of RAM it should not access
- can we avoid memory reservation in coreboot tables?
    - yes, by generating and updating EPT (*Extended Page Table*) at runtime as well as implementing memory management functions for VMs
    - that approach would have significant impact on hypervisor size and performance
    - not suitable for embedded hypervisor applications

Hypervisor as coreboot payload



Demo time

Future improvements

- include Bareflank build in coreboot
- flexible partitioning is important, so some form of configuration should be introduced
- memory map management should be improved
- we should update Bareflank code base (we base on early 2019 version)
- AMD SVM support can provide additional value

Q&A