kHypervisor

Kelvin Chan
Game Security Researcher
Tencent

Who am I?

Kelvin Chan

Game Security Researcher, Tencent
Windows Kernel and low level stuff enthusiast
Game Security enthusiast
Twitter: @kelvin1272011
Email: kelvinchan@tencent.com / kelvin1272011@gmail.com

Agenda

- Project Motivation
- Common Hypervisor
- Nested Virtualization Internals
- Future

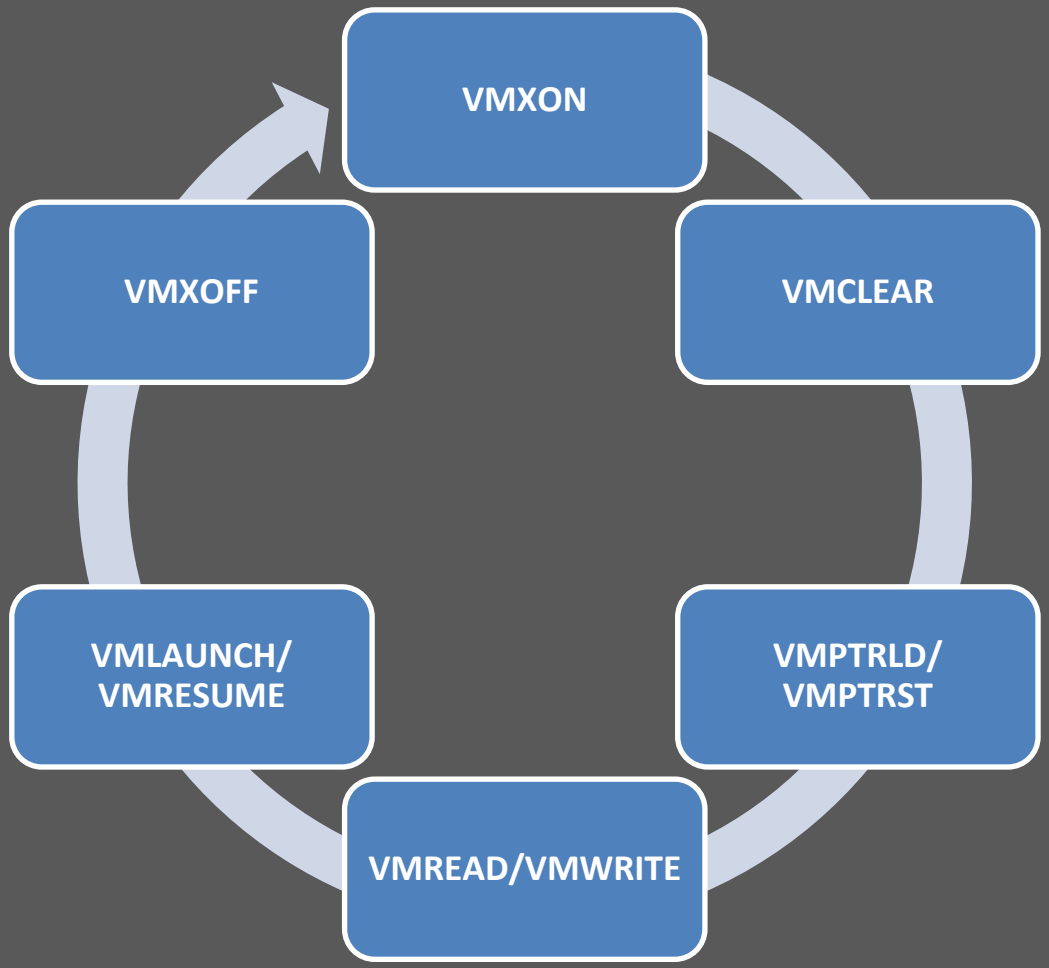
Motivation

- Key objective is learning
- Providing a detailed and properly documented project
- Minimalist and straightforward code based
- Focusing on Intel VT-x virtualization
- Building a comprehensive hypervisor debugging framework
- Portable, Modifiable, Simple

Common Hypervisor

- Common hypervisors are too complex for beginners
- Nested virtualization is unsupported
  - VirtualBox
- Heavy weight, hard to understand.
  - KVM, BOCHS
- Closed-Source
  - VMWare (WorkStation, ESX), Microsoft Hyper-V

Hypervisor Lifecycle



How does it work?

- Overview
- Virtualization
  - VM instructions
  - VMExit
  - VMEntry
  - VMCS
  - EPT
- Goals
- Use cases

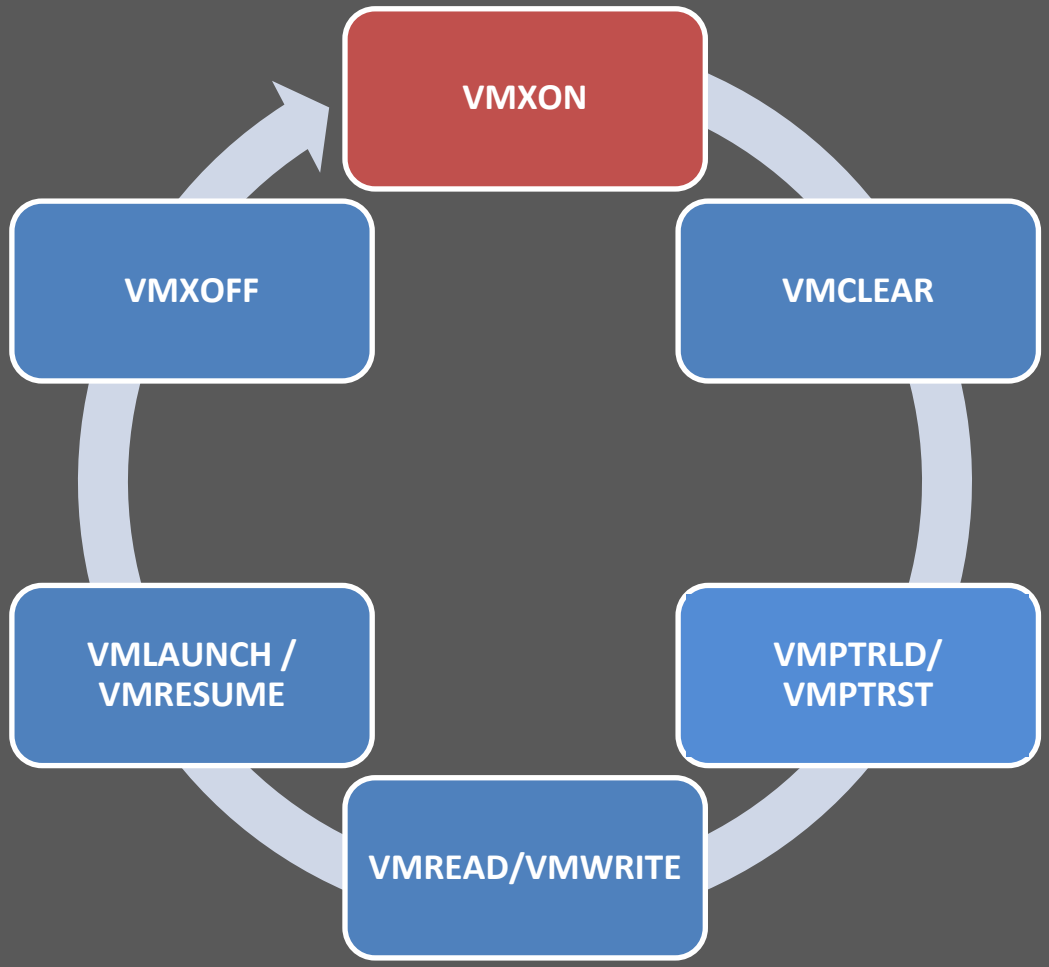
Overview

- Extending the work of HyperPlatform, a simple VT-x framework by Satoshi Tanda.
  - https://github.com/tandasat/HyperPlatform
- Simple
  - The Nested Virtualization part is implemented in less than 3500 line of commented code.

Virtualized instructions

- VmxVmxonvirtualize
- VmxVmxoffvirtualize
- VmxVmclearvirtualize
- VmxVmptrldvirtualize
- VmxVmreadvirtualize
- VmxVmwritevirtualize
- VmxVmlaunchvirtualize
- VmxVmresumevirtualize
- VmxVmptrstvirtualize
- VmxVMExitvirtualize

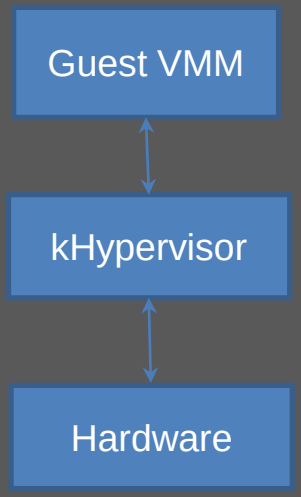
Hypervisor Lifecycle



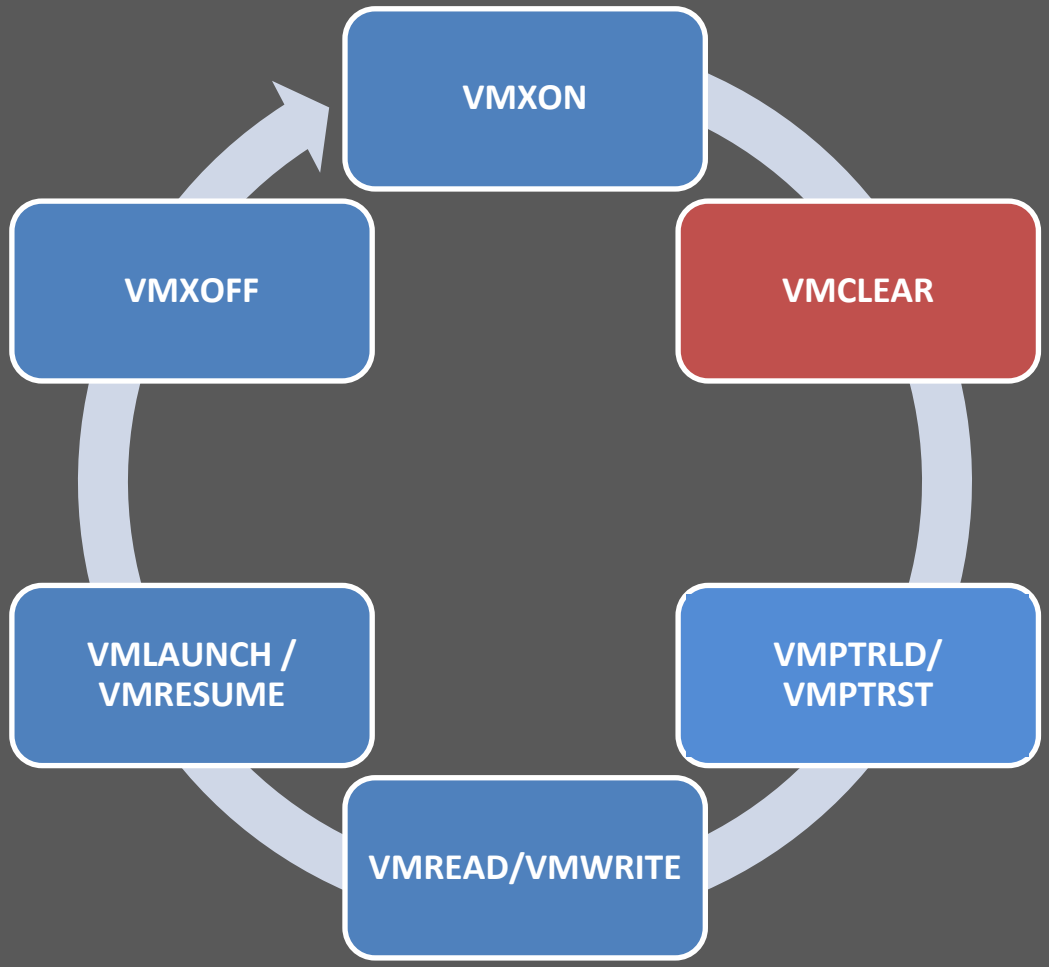
VMXON / VMXOFF

Trap these instructions and do following work:

- Some regular parameter check
- Initialize the Virtual CPU context and set the VMX mode
- Save the current virtual machine control structure (VMCS)
  - VMCS0-1
    - L0 context for L1 use

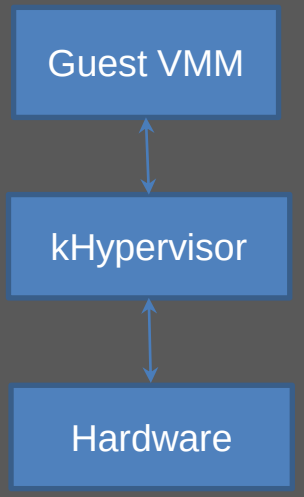


Hypervisor Lifecycle

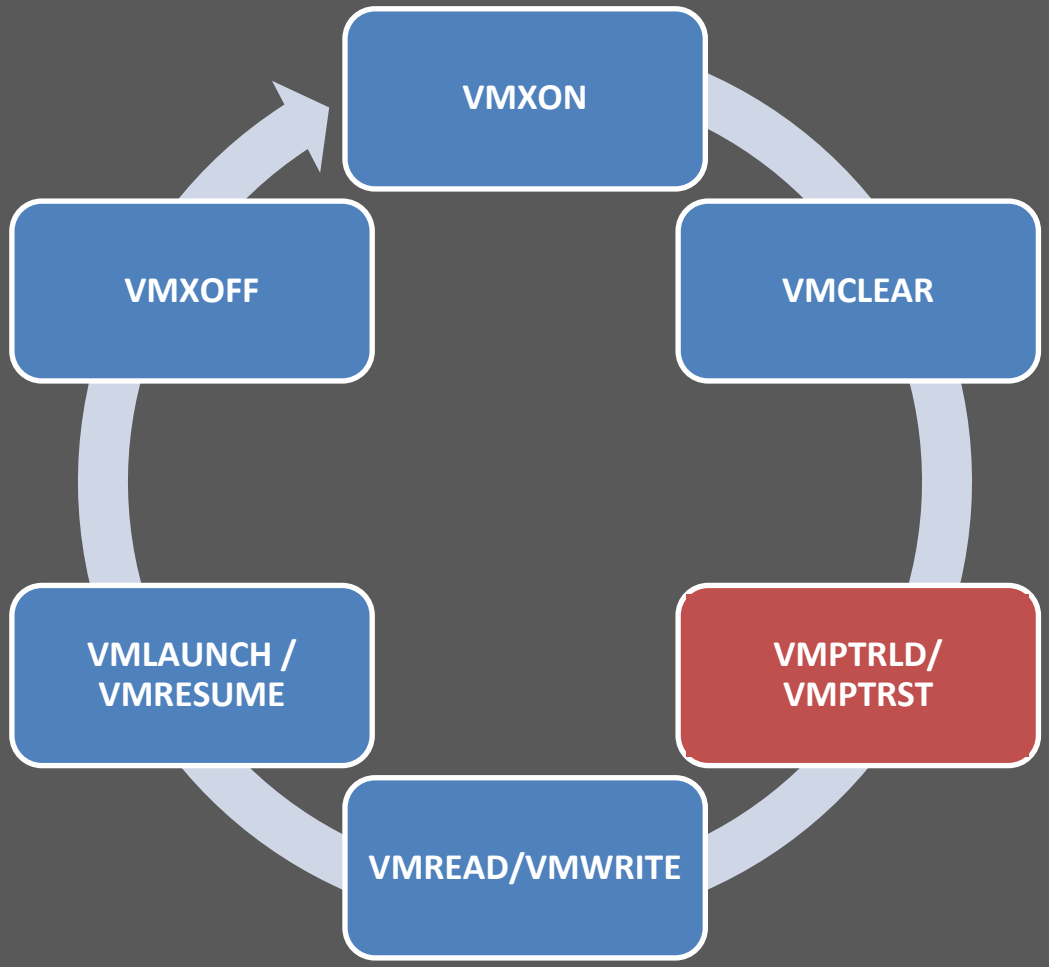


VMCLEAR

- Clear the VMCS with the original VMCLEAR instruction
- Ensure that VMCS0-2 can be loaded into CPU
- Is executed before launching the Guest OS.



Hypervisor Lifecycle

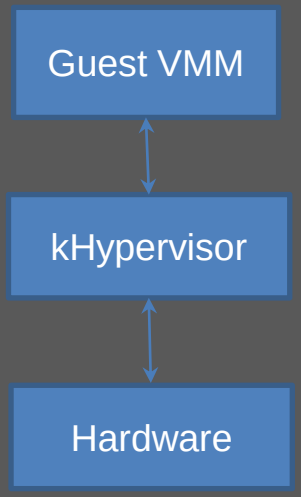


VMPTRLD/ VMPTRST

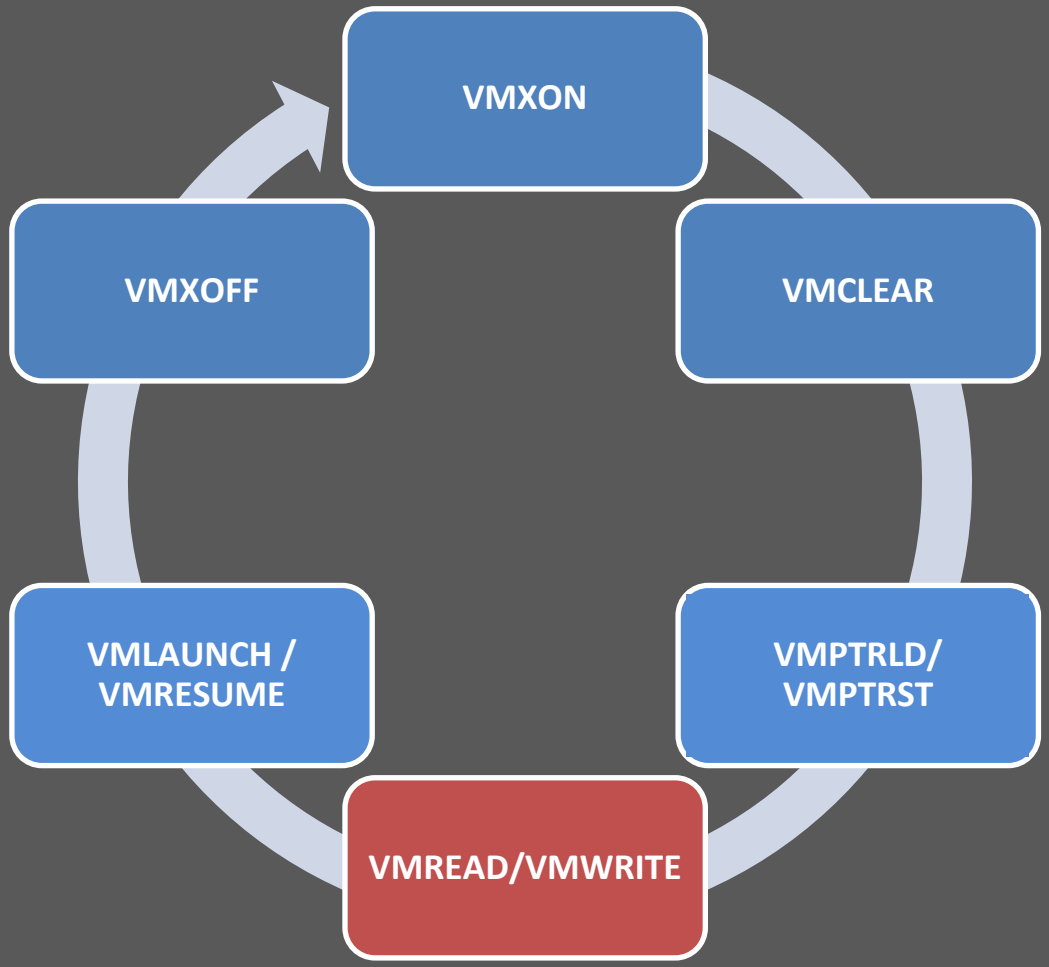
VMPTRLD/VMPTRST instructions are responsible for loading the VMCS into the CPU

- kHypervisor saves the VMCS (VMCS1-2) from the input parameter to the current CPU
- kHypervisor create the new VMCS (VMCS0-2) for the current virtual CPU

- Results in two distinct VMCS:
  - VMCS1-2 and VMCS0-2

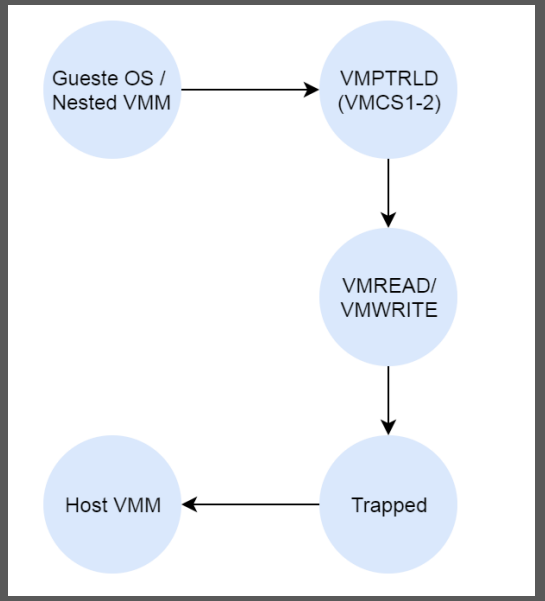


Hypervisor Lifecycle



VMWRITE / VMREAD

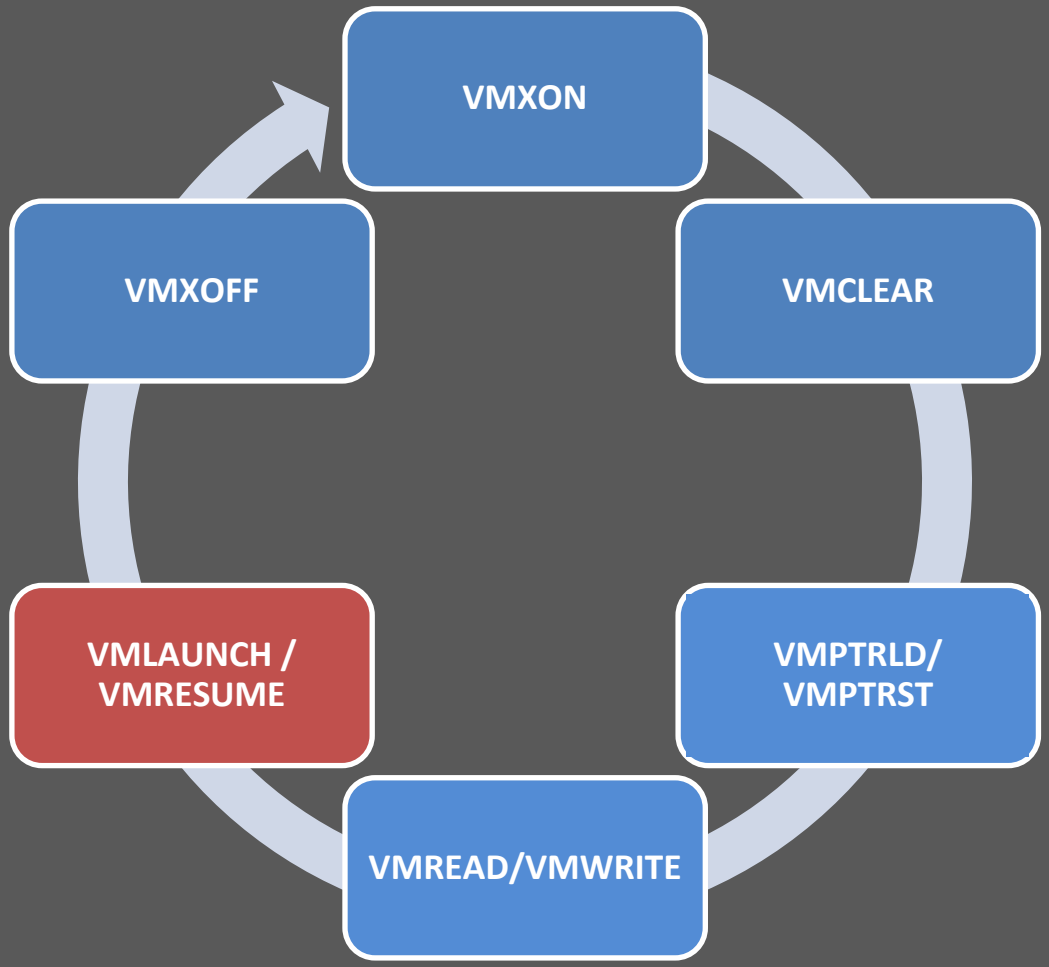
- Guest VM is not aware that it is being virtualized
- After VMPTRLD execution, the Guest try to read/write its VMCS 1-2
- Trap the R/W instructions to intercept such events
- Direct R/W on the VMCS 1-2 memory space
- The memory layout VMCS 1-2 can be customized by kHypervisor
- Decode the parameter of R/W functions
- Use simple hash function and find the offset and perform R/W




```
VOID
VmcsVmRead64(
        _In_ VmcsField Field,
        _In_ ULONG_PTR base,
        _In_ PULONG64 destination
)
{
        ULONG_PTR offset = GetVMCSOffset((ULONG64)Field);
        *destination = *(PULONG64)(base + offset);
}
```

```
VOID
VmcsVmRead32(
        _In_ VmcsField Field,
        _In_ ULONG_PTR base,
        _In_ PULONG32 destination
)
{
        ULONG_PTR offset = GetVMCSOffset((ULONG64)Field);
        *destination = *(PULONG32)(base + offset);
}
```

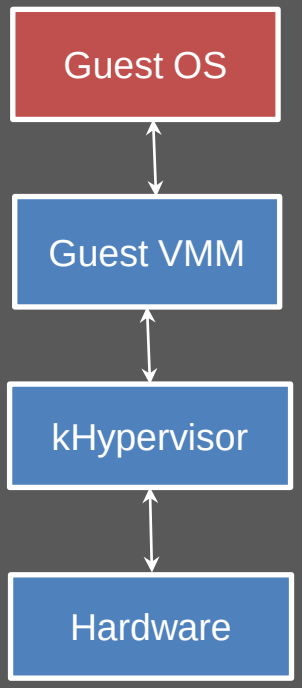
Hypervisor Lifecycle



VMLAUNCH/ VMRESUME

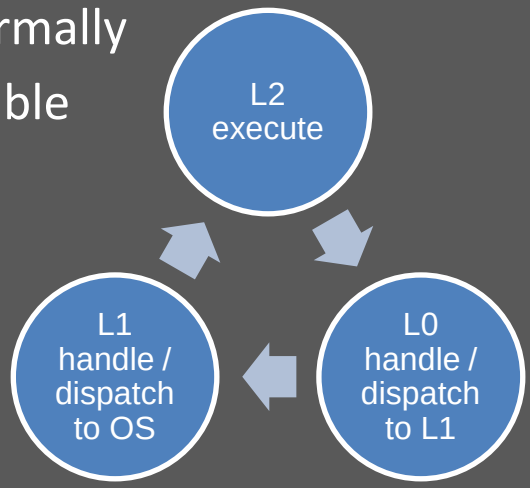
Launch the OS and VMM in the virtualized environment:

- Argument check
- Merging VMCS
- VMEntry virtualization



Expected Results

- L2 to be successfully launched and to work normally
- Events from L2 to be trapped as much as possible
- Everything to be under L0 control
  - Guest VMM(L1) will be trapped by VMCS0-1
  - Guest OS(L2) will be trapped by VMCS0-2



VMentry virtualization

Level 1 Guest VMM will execute VMEntry with the assistance of Level 0 Hypervisor

Core Concept

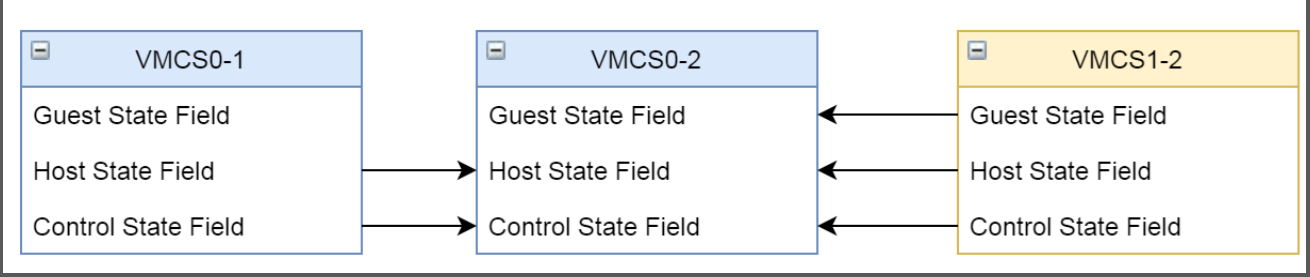
- Leveraging VMLAUNCH / VMRESUME to switch back to guest mode of physical CPU, and control the execution path

- Implementation

  Control the flow by VMCS, this provide different destination for the guest (included Guest VMM and OS)

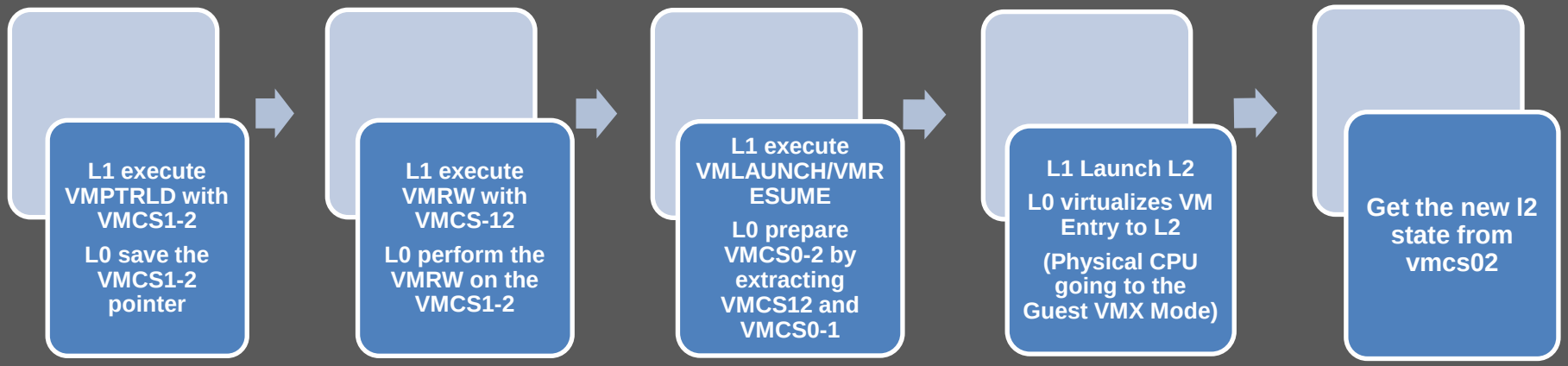
VMCS0-2 Layout

- VMCS0-2 Control State contains both VMCS0-1 and VMCS1-2 control states.
  - VMCS0-2 includes everything L1 and L0 requires.
- VMCS0-2 Host State contains both VMCS0-1 and VMCS1-2 host states
  - They are used to ensure L2 can be correctly trapped
- VMCS0-2 Guest Field is used for launch L2 whatever L1 wants to create



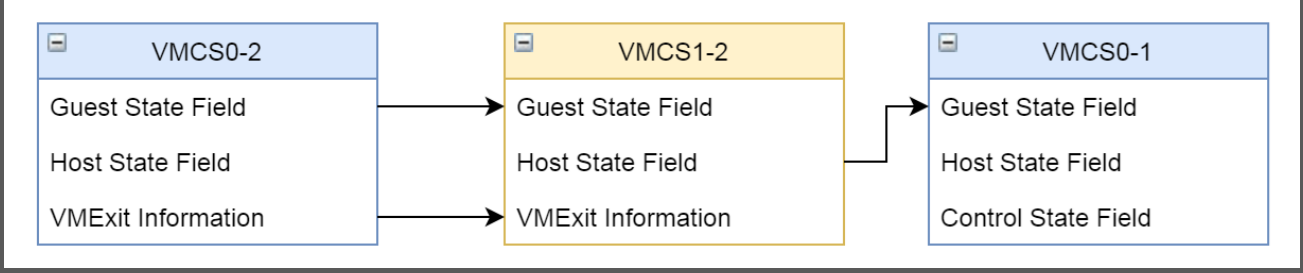
Merging VMCS

- VMCS0-1 from Level 0 to launch Level 1
- VMCS1-2 from Level 1 to Level 2. This won't be loaded but is required by the guest
  - L1 always think it is being used
- VMCS0-2 from Level 0 to Launch level 2



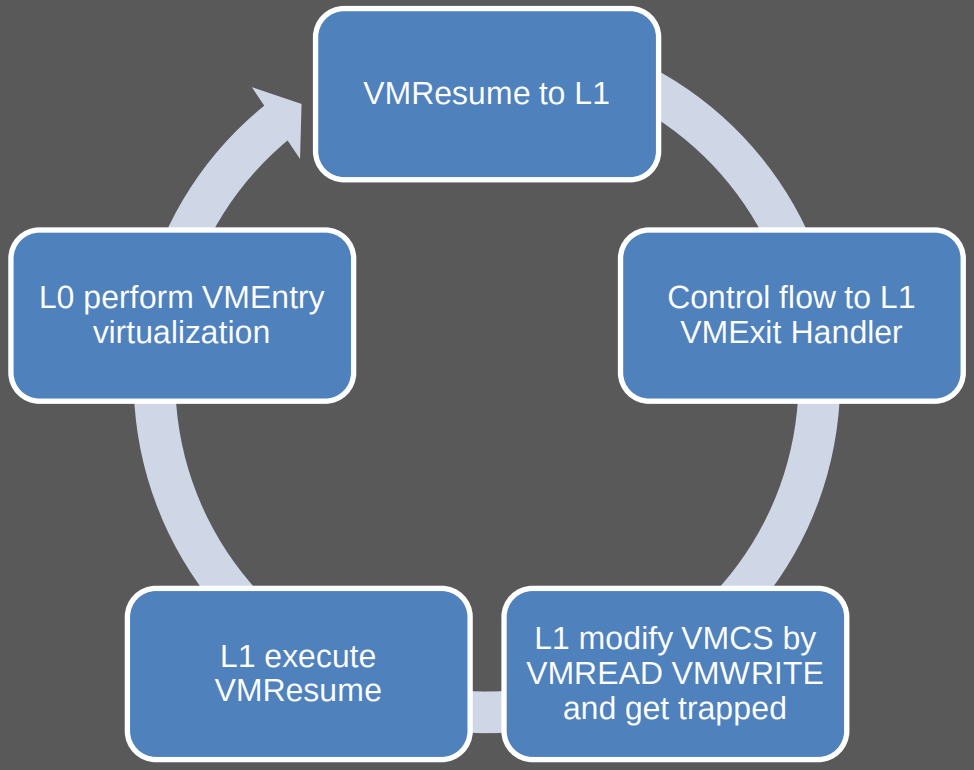
VMExit virtualization

- Core Concept
  - Leverage VMRESUME to perform VMEXIT virtualization
- Implementation
  - Control the flow by VMCS, this provide different destinations for the guest (included Guest VMM and OS)
  - Fake “VMExit” but L0 hands off the control to L1 along with the VMX mode change (Guest Mode)
  - The fake “VMExit” will trigger the L1 VMM trap again, even though L1 think it is already root mode.
  - Finalizing the fake VMExit by trapping into L0 again (VMRESUME)
- VMCS01 Layout
  - Saved copy of the VMCS0-2 Guest State into VMCS1-2
  - Saved copy of the VMCS0-2’s exception related context into VMCS1-2
  - Loading the L1 VMExit handler in VMCS0-1 Guest Rip field along with its stack, rlfags, etc.



VMExit virtualization

- The processor control flow turn to L1 VMExithandler.
- L1 tries to read / write the information from/to VMCS1-2 which triggers VMExit from L1 again



EPT virtualization

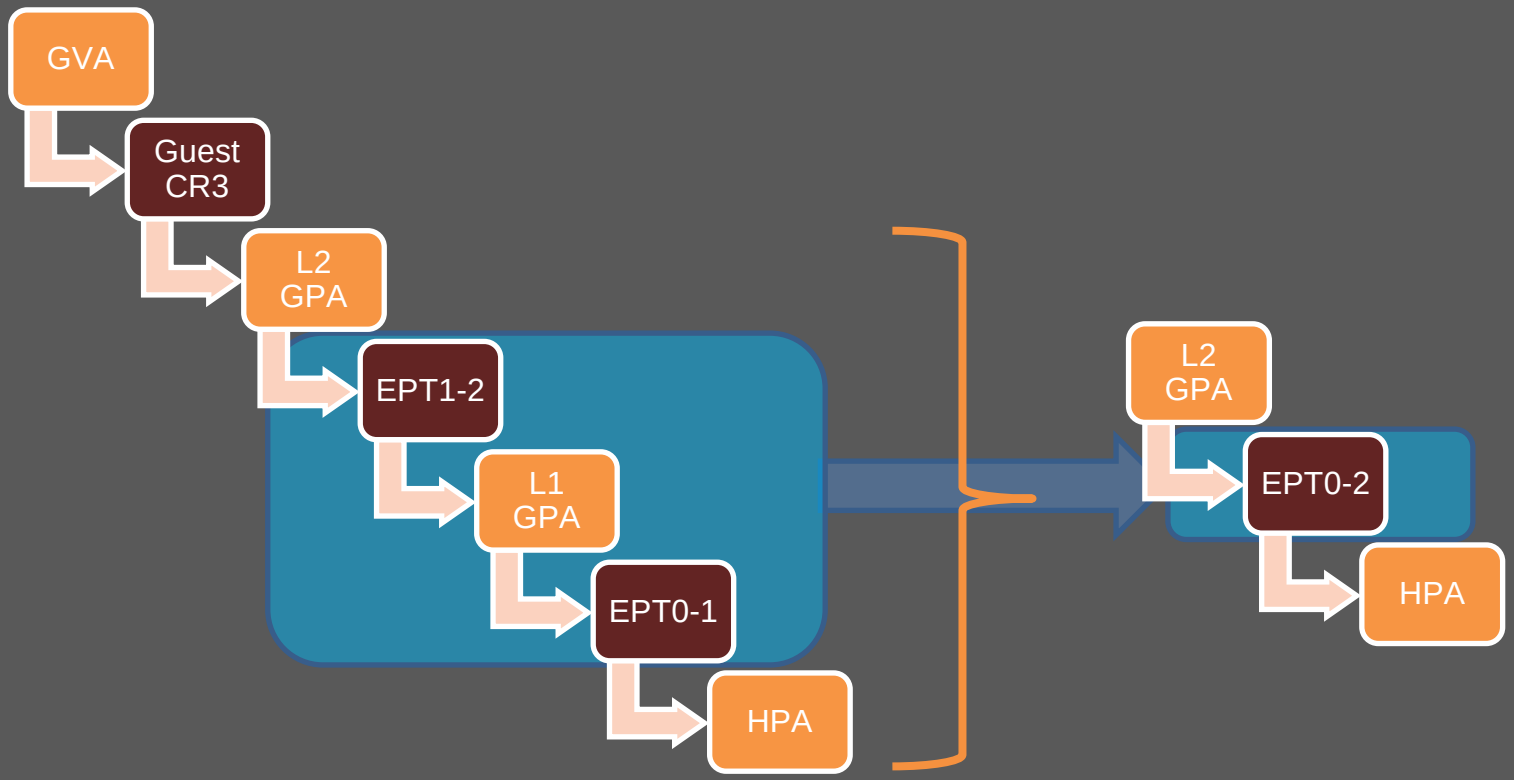
EPT translates Guest Physical Address (GPA) to Host Physical Address (HPA)

Processors only accept 2-level address translation

Guest EPT are monitored by Level 0

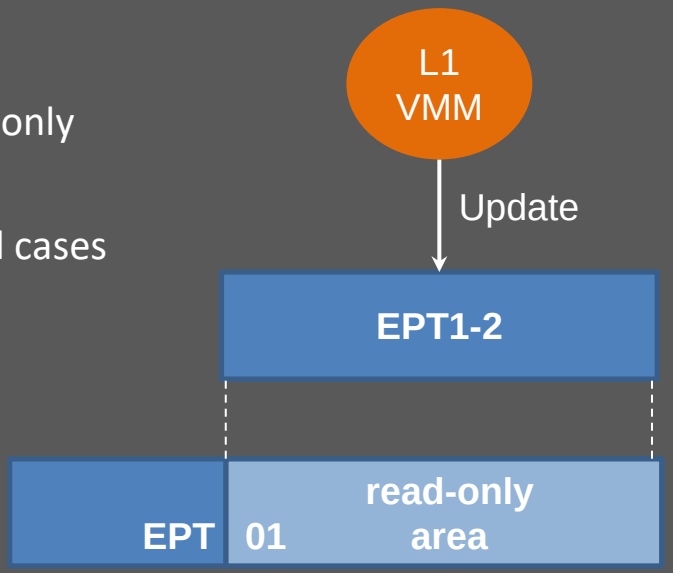
Use Monitor Trap Flag (MTF) for trapping any modification

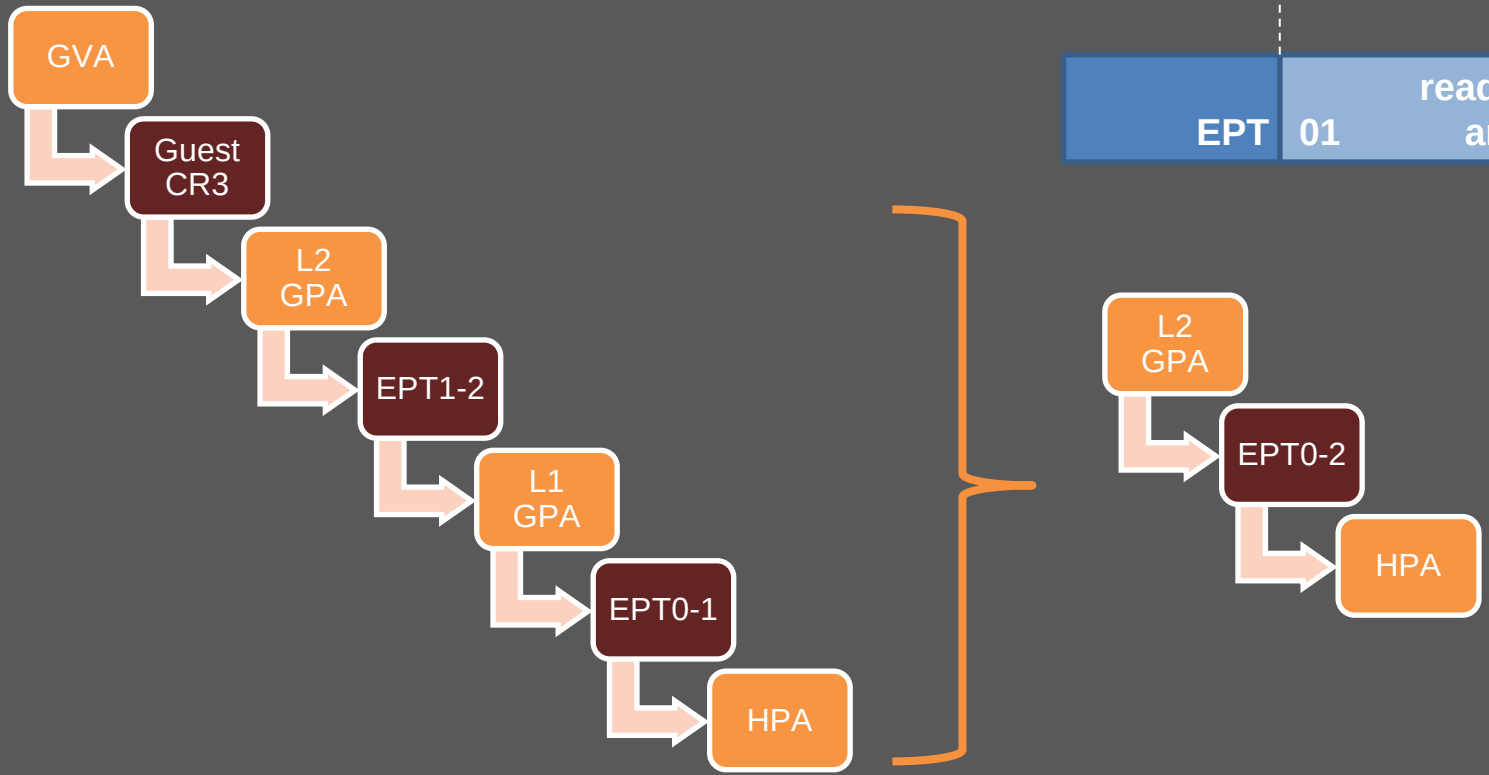
EPT0-2 = EPT1-2 + EPT0-1, give it to the processor which performs the translation for us



EPT virtualization

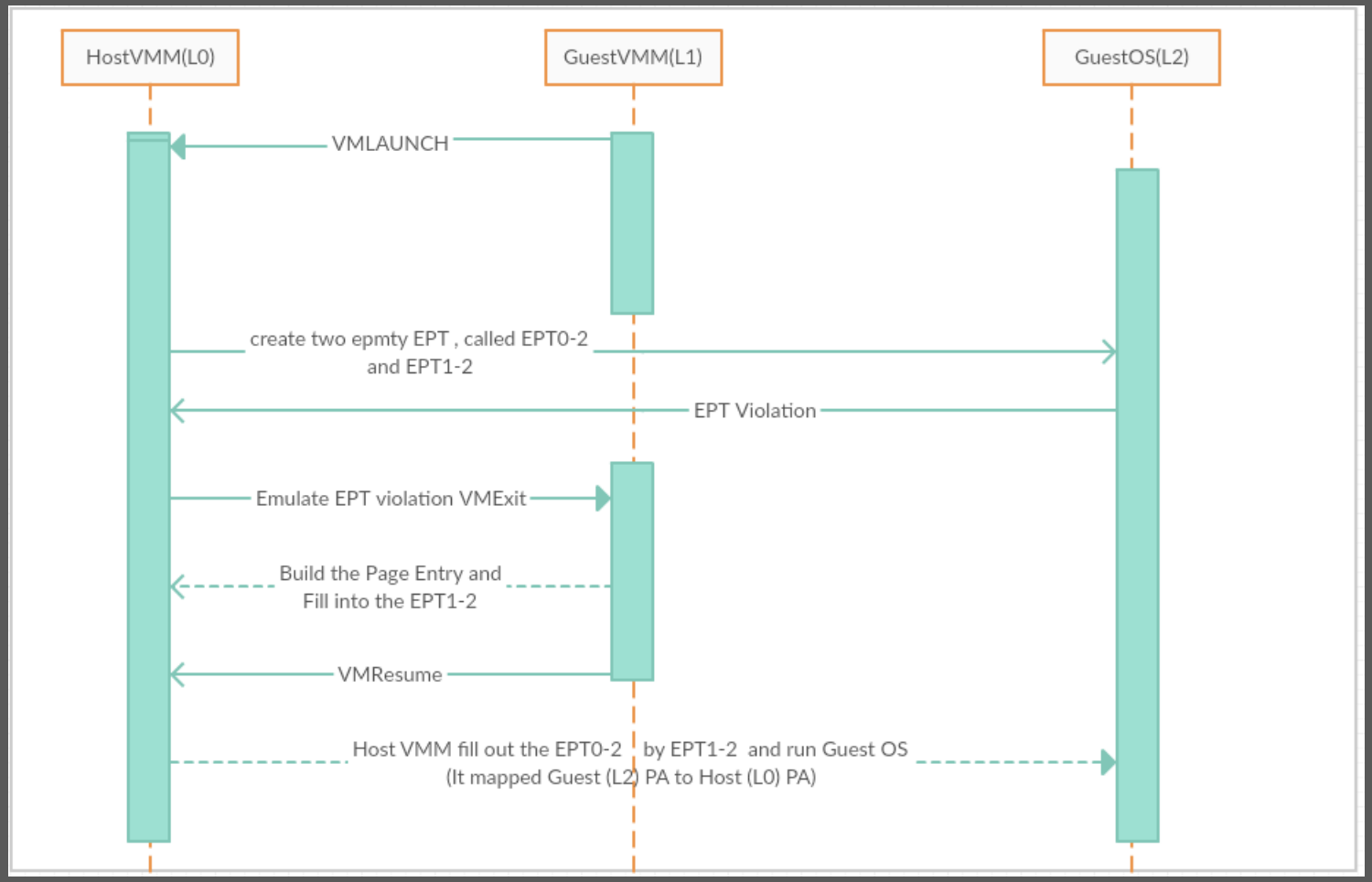
- The page entry of EPT1-2 in EPT0-1 is always mark as read-only
- Any access to EPT1-2 from L1 will causes EPT violation
- Set Monitor Trap Flag (MTF) to follow up with those special cases
- EPT0-2 can be updated by accessing EPT1-2 after a write
- Reset the page attributes and reload the EPT0-2





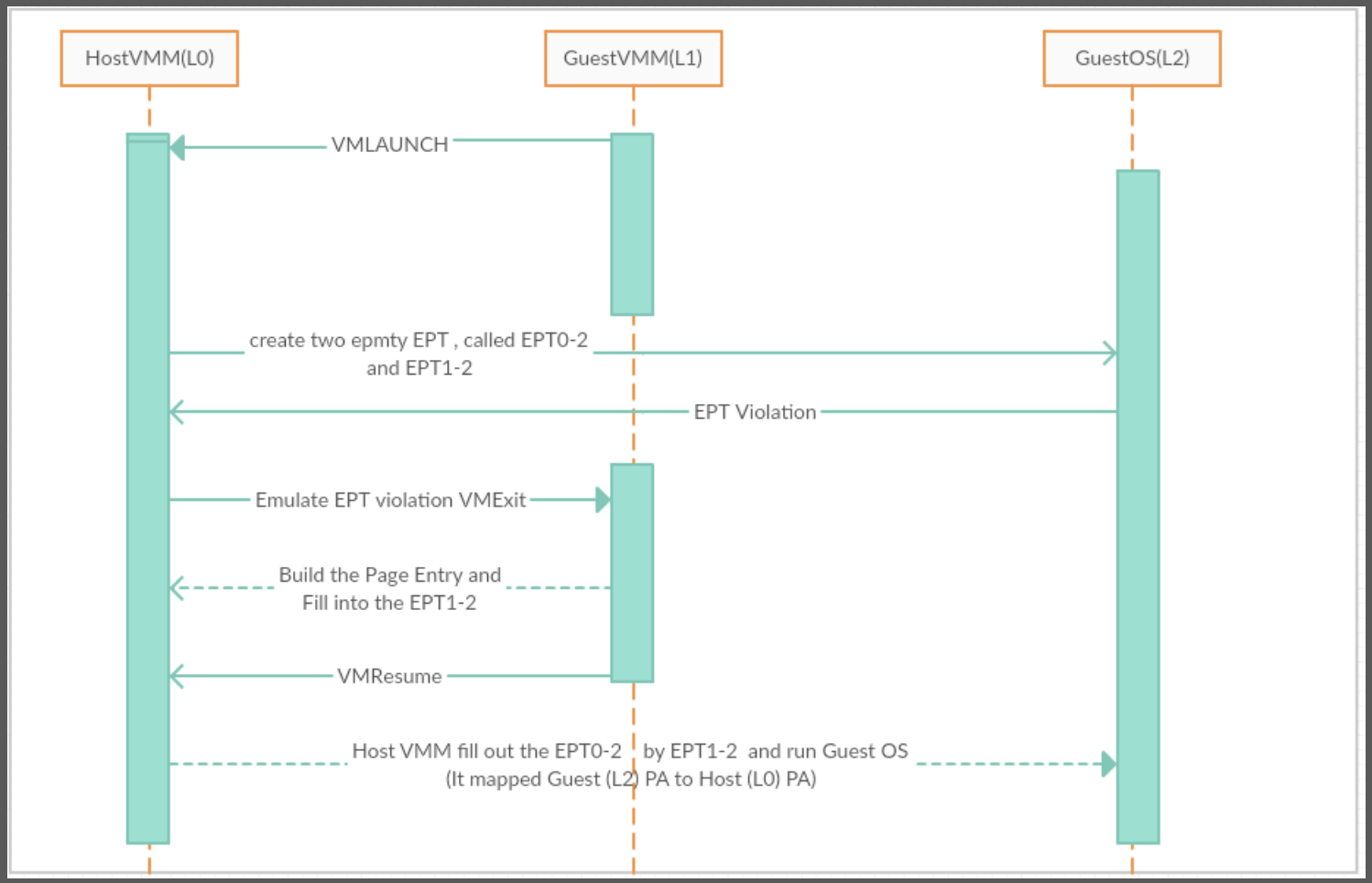
EPT virtualization

- EPT-on-EPT, build entry on-the-fly

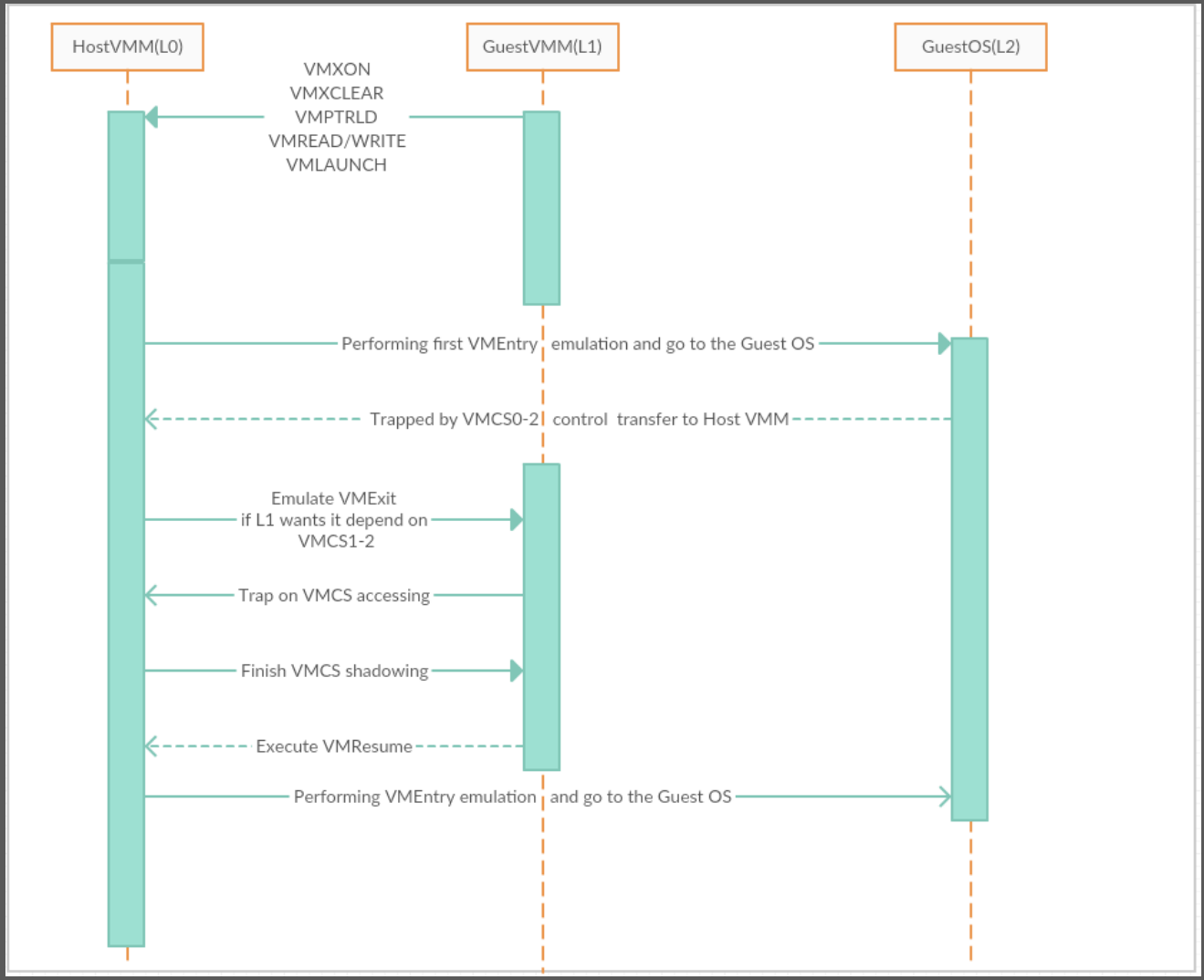


EPT virtualization

- EPT-on-EPT, build entry on-the-fly

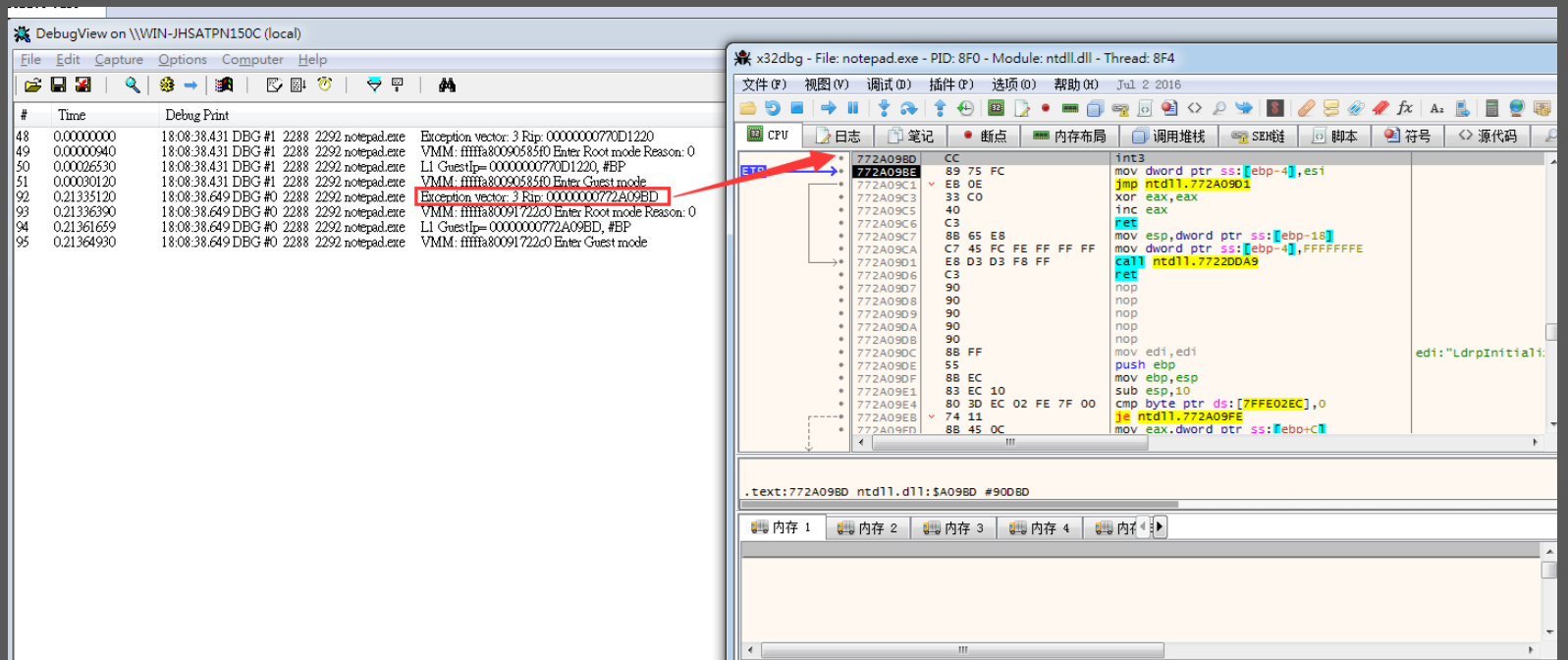


Full Picture - Review



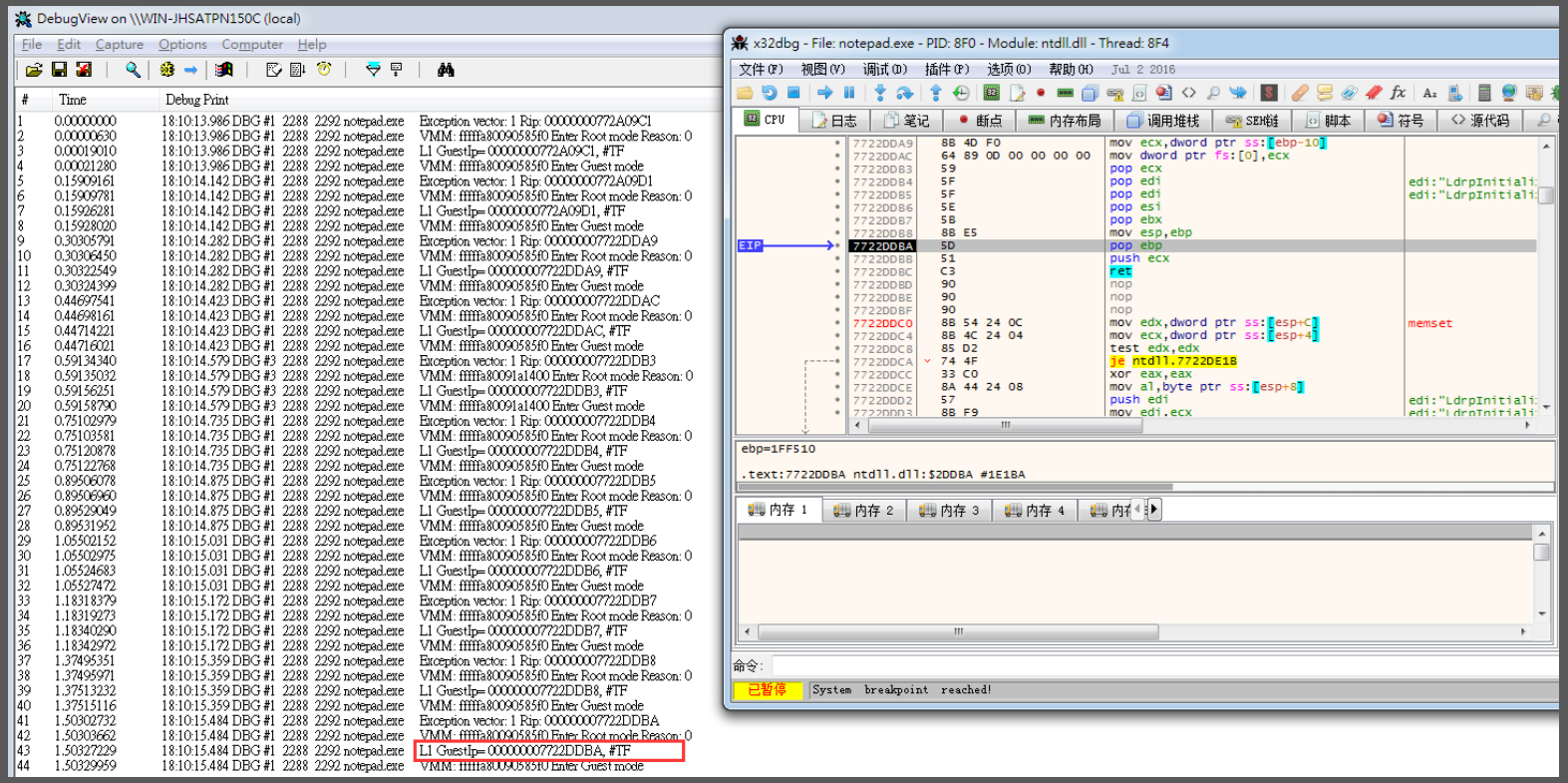
Use Case

- Analysis of hypervisor-based rootkits
  - The hypervisor offers an easy way to trigger exceptions / interrupts
- kHypervisor offers the ability to monitor the redirection of exceptions and interrupts
  - We are transparent
  - Rootkits are still functioning as expected.



Use Case

- Monitoring the single stepping exceptions along with hypervisor-based rootkit.
- Monitoring shadow walker



Nested debugging

- Larger scope of debugging!
- Nested Virtualization Debugging
- Host VMM can take over all of the exceptions and interrupts.
- VirtIce provided the following functions:
  - VMExit / VMEntry breakpoint and tracing
  - Put the Hyper-malware sample into the debugger

- To be continued…

More

- Provide VMExit tracing features
- Enabling on-the-fly Hypervisor debugging
- Malware Analysis

Resources

- Github
  - https://github.com/Kelvinhack/kHypervisor
- How does Nested Virtualization Works? (February 2018)
  - https://kelvinhack127.blogspot.com/2018/02/how-does-nested-virtualization-works.html

Further Research

- APIC virtualization
- Windows Debugging features
- Nested Hypervisor debugging

Conclusion

- Nested Virtualization Debugging is important for the security researchers especially because of the rise of hypervisor-based malwares and rootkits.
  - Those are impossible to debug with traditional debugging, but Nested Virtualization Debugging is a solution to that problem.

Thanks

- My teammates
- Satoshi Tanda @standa_t
  - Author of HyperPlatform