Linux Virtualization Based Security (LVBS)

Angelina Vu

Motivation

- Linux Kernel vulnerabilities have been steadily rising and getting exploited in the wild
- Our goal is to:
  1. Harden the kernel by enforcing protections, which cannot be turned off by a malicious kernel
  2. Ensure that critical system assets (keys, critical kernel data structures) remain untampered, even if the kernel gets compromised

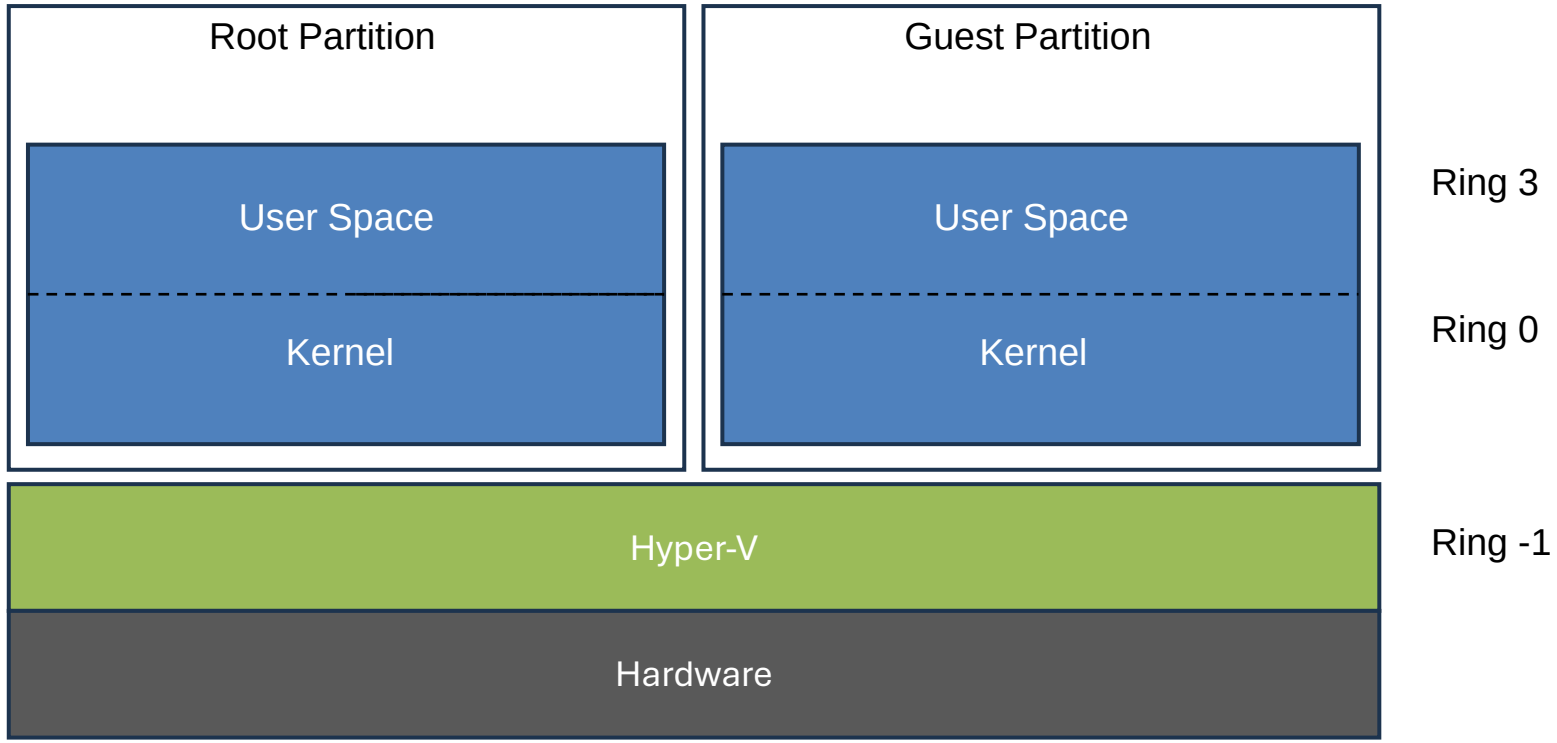
Linux Virtualization-based Security (LVBS)

- Inspired by Windows Virtualization-based Security (VBS)
  - Uses hypervisor and hardware virtualization to protect guest OS
- Open-source architecture
  - Hardware agnostic
  - Hypervisor agnostic

Hardware Requirements

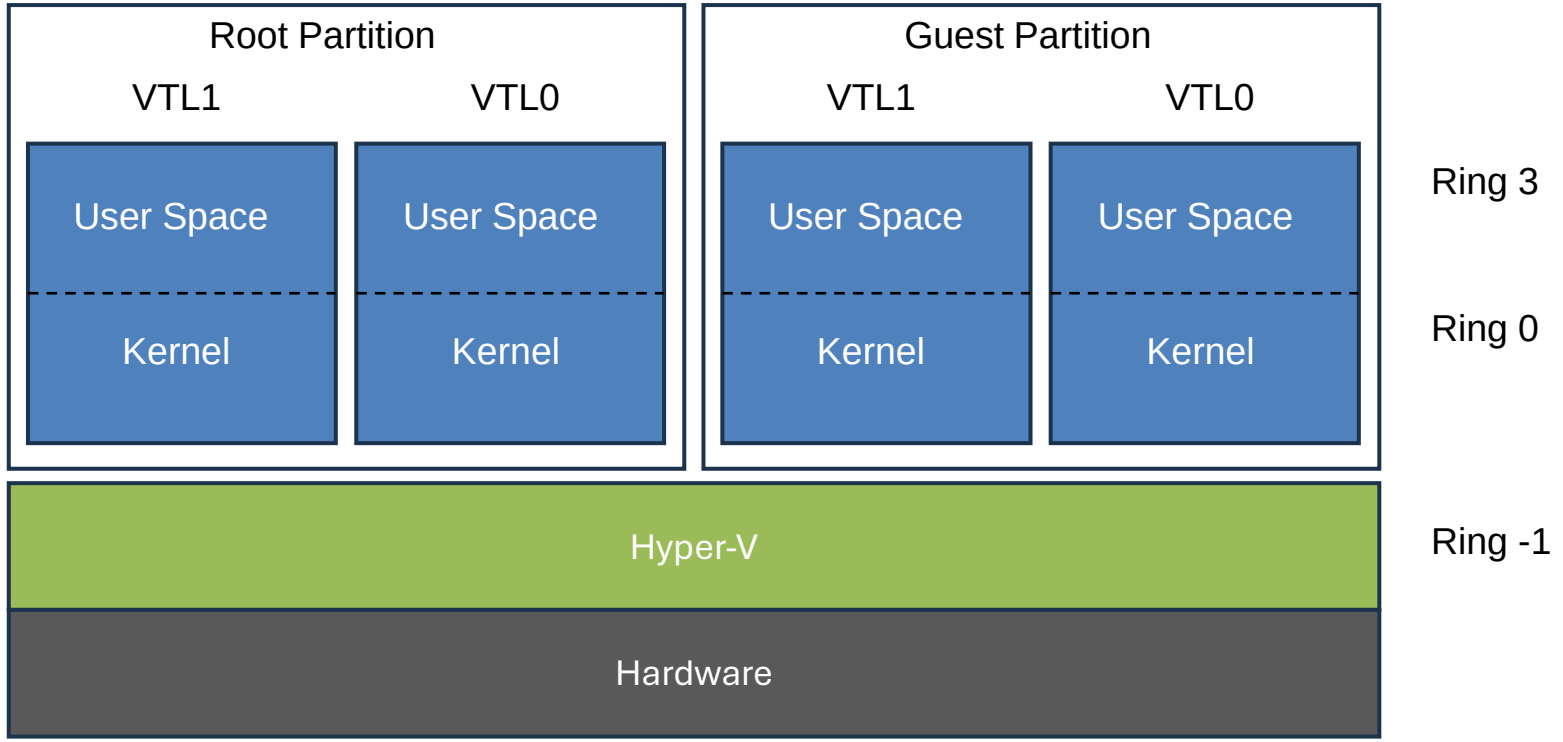
- Second Level Address Translation (SLAT)
  - Manage VM memory and add a secondary complementary layer of permissions only controlled by the hypervisor
  - Intel’s EPT, AMD RVI/NPT
- Mode Based Execution Control (MBEC)
  - Split user and kernel execute permissions

Hyper-V


Root Partition
Guest Partition
User Space
Kernel
User Space
Kernel
Hyper-V
Hardware
Ring 3
Ring 0
Ring -1

Hyper-V's Virtual Secure Mode (VSM)

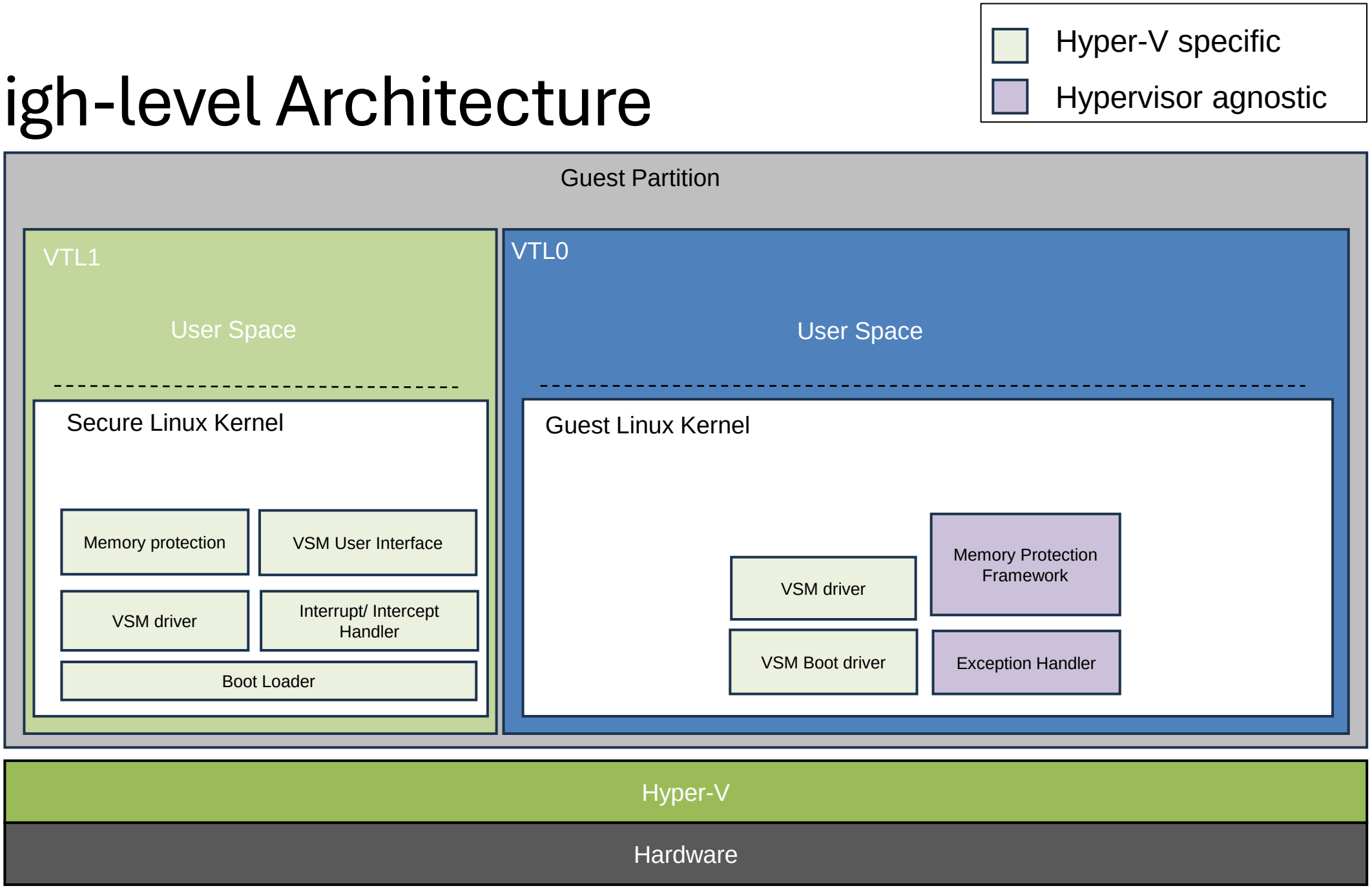
- Introduces separate execution environments within a partition, called Virtual Trust Levels (VTLs)
- Higher VTLs more privileged than lower VTLs



VSM Features

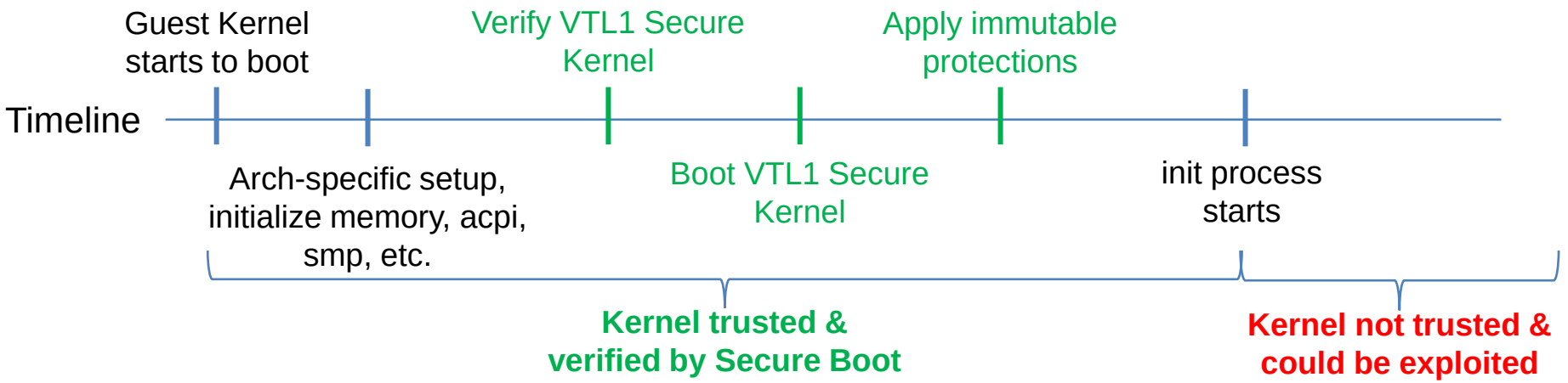
- Virtual Processor state isolation
  - VPs maintain separate, per-VTL state
    - Each VTL defines a set of private VP registers
- Memory access hierarchy and protection
  - Each VTL maintains a set of guest physical memory access protections
  - Higher VTLs can impose memory restrictions to lower VTLs
  - Access protections implemented by hypervisor via Extended Page Table (EPT) or Second Level Page Table (SLT)
- Virtual Interrupt and Intercept handling
  - Each VTL has its own interrupt subsystem (local APIC). This ensures that higher VTLs process interrupts without interference from lower VTLs.

High-level Architecture



Threat Model

- Goal: Protect kernel from userspace attacker exploiting kernel vulnerability
- Defence in depth
- Hypervisor within TCB
- Guest kernel trusted from boot (verified by Secure boot) until first unverified userspace process (i.e., init)



VTL0 – VTL1 Interface

- Currently, no process runs in VTL1 continuously
- VTL1 is entered when a VP switches from VTL0 to VTL1. This can happen via:
  1. VTL calls (synchronous)
     - Guest kernel (VTL0) makes an explicit hypercall to invoke VTL1

Category	VTL Call Opcode
VTL1 Boot	VSM_VTL_CALL_FUNC_ID_ENABLE_APS_VTL VSM_VTL_CALL_FUNC_ID_BOOT_APS
Apply protections	VSM_VTL_CALL_FUNC_ID_PROTECT_MEMORY VSM_VTL_CALL_FUNC_ID_LOCK_CR VSM_VTL_CALL_FUNC_ID_SIGNAL_END_OF_BOOT
Load VTL0 Data	VSM_VTL_CALL_FUNC_ID_LOAD_KDATA VSM_VTL_CALL_FUNC_ID_COPY_SECONDARY_KEY
Module Loading/Unloading	VSM_VTL_CALL_FUNC_ID_VALIDATE_MODULE VSM_VTL_CALL_FUNC_ID_FREE_MODULE_INIT VSM_VTL_CALL_FUNC_ID_UNLOAD_MODULE

VTL0 – VTL1 Interface

2. Secure interrupts (asynchronous): If an interrupt is received for a higher VTL, the VP will enter the higher VTL
   - After VTL1 has booted, timer interrupts are disabled while the VP is in VTL0. This greatly reduces jitter.
3. Secure intercepts (asynchronous): When VTL0 violates VTL1 protections, the VP that triggered the fault enters VTL1

Register Pinning

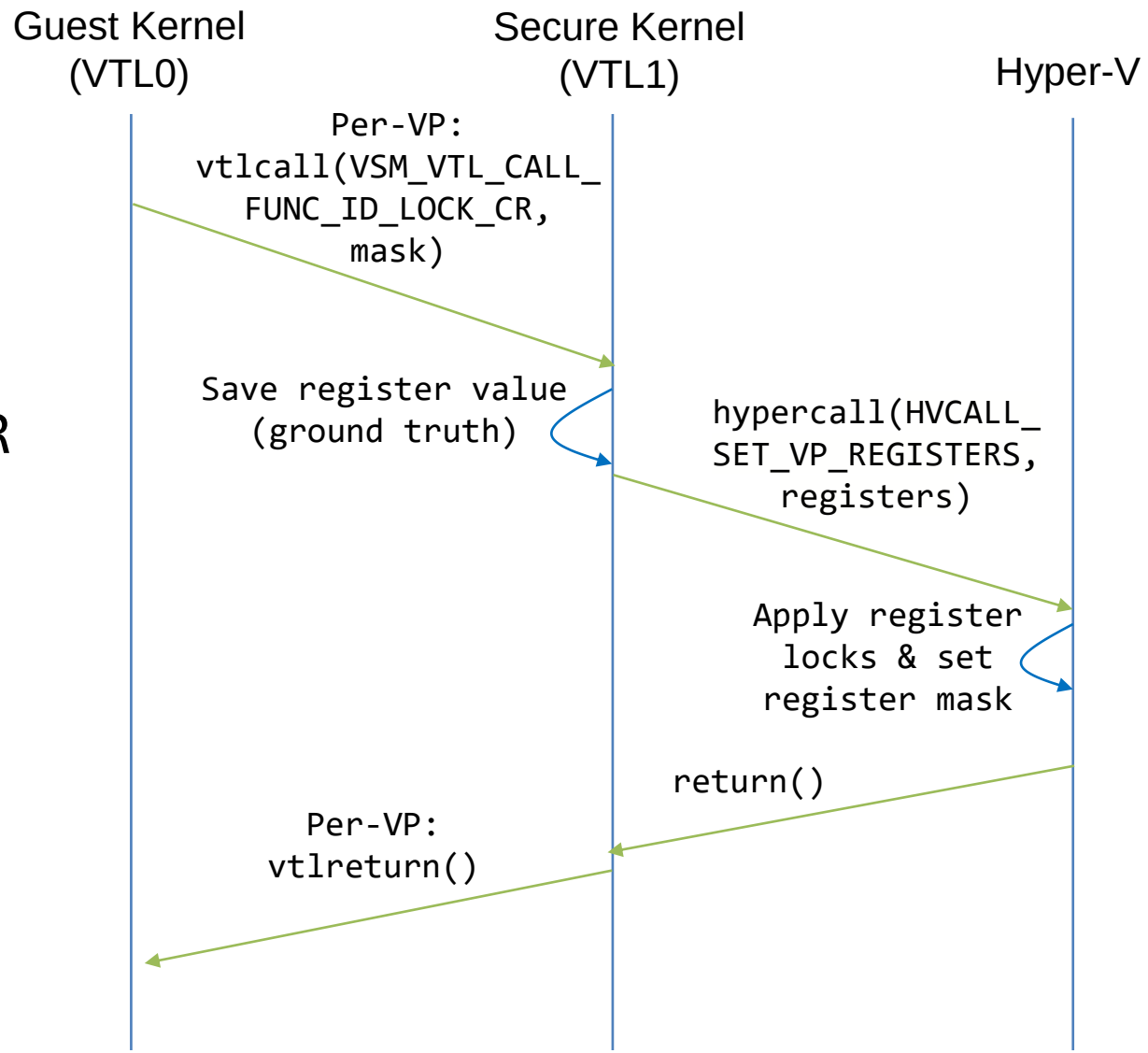
- Hyper-V supports intercepting access to a number of registers
- 2 sets of registers
  - Block writes altogether
  - Allow writes, if the value is the same as pre-init

Action	Registers
Block write & raise GP fault in VTL0	GDTR, IDTR, LDTR, TR
Allow write if new value is the same as pre-init. Otherwise, block write & raise GP fault in VTL0	CR0, CR4, LSTAR, STAR, CSTAR, APICBASE, EFER, SYSENTER_CS, SYSENTER_ESP, SYSENTER_EIP, SYSCALL_MASK

```
/*  CR Intercept Control */
union hv_cr_intercept_control {
    u64 as_u64;
    struct {
        u64 cr0_write               : 1;
        u64 cr4_write               : 1;
        u64 xcr0_write              : 1;
        u64 ia32miscenable_read     : 1;
        u64 ia32miscenable_write    : 1;
        u64 msr_lstar_read          : 1;
        u64 msr_lstar_write         : 1;
        u64 msr_star_read           : 1;
        u64 msr_star_write          : 1;
        u64 msr_cstar_read          : 1;
        u64 msr_cstar_write         : 1;
        u64 msr_apic_base_read      : 1;
        u64 msr_apic_base_write     : 1;
        u64 msr_efer_read           : 1;
        u64 msr_efer_write          : 1;
        u64 gdtr_write              : 1;
        u64 idtr_write              : 1;
        u64 ldtr_write              : 1;
        u64 tr_write                : 1;
        u64 msr_sysenter_cs_write   : 1;
        u64 msr_sysenter_eip_write  : 1;
        u64 msr_sysenter_esp_write  : 1;
        u64 msr_sfmask_write        : 1;
        u64 msr_tsc_aux_write       : 1;
        u64 msr_sgx_launch_ctrl_write   : 1;
        u64 reserved                : 39;
    };
} __packed;
```

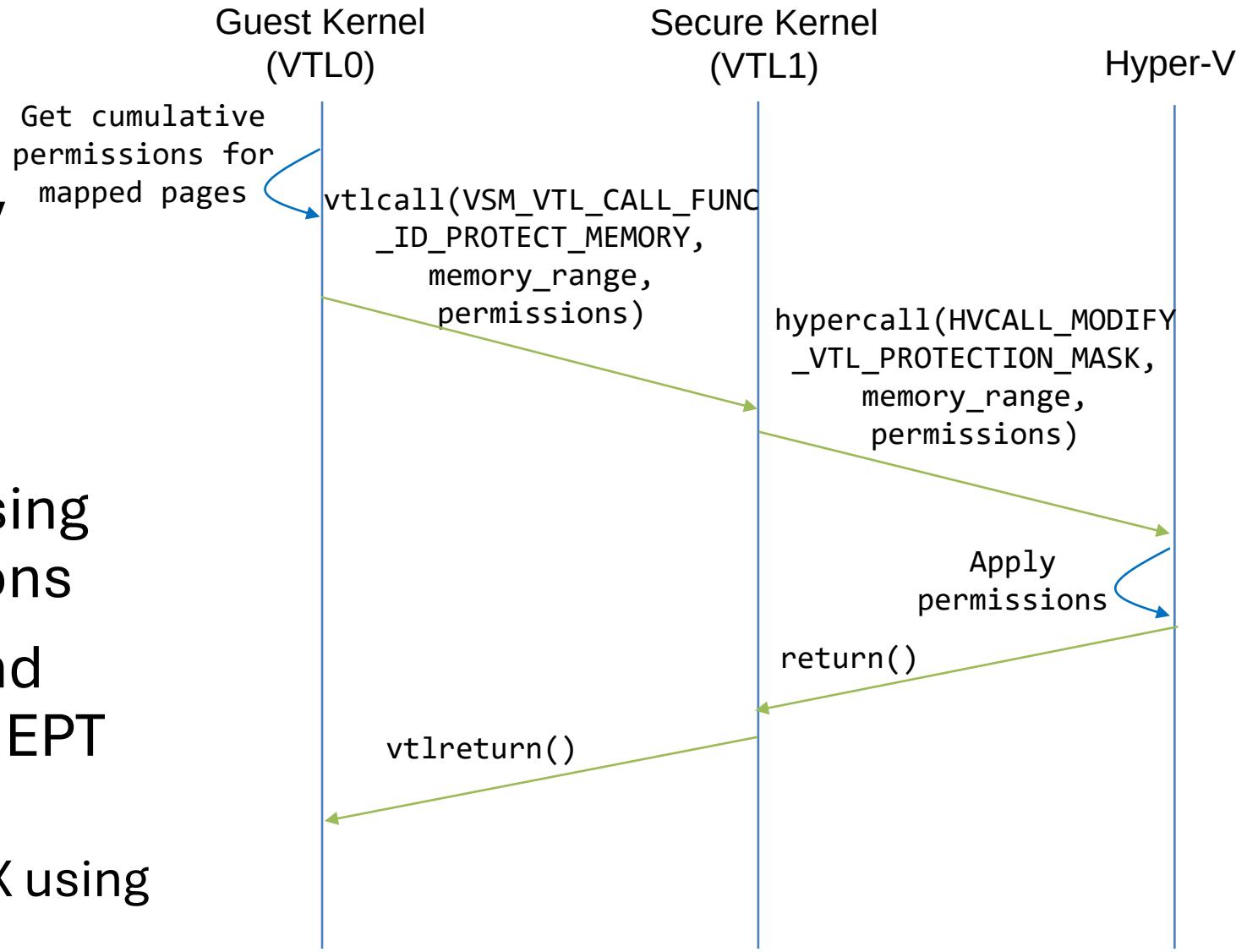
Register Pinning

- VTL0 sets security policy of control registers (CRs) to protect
- CRs are per VP
  - For each VP, VTL0 does a LOCK_CR VTL call
- VTL1 sends hypercall request to lock CRs
- Hypervisor applies protection
- Unallowed write request to monitored register, injects intercept to VTL1



Memory Protection

- Guest kernel uses memory protection framework
  - Hypervisor agnostic
- Walk mappings to get cumulative permissions using guest page table permissions
- Pass guest kernel pages and permissions to VTL1 to set EPT permissions
  - Read, write, execute (uX / kX using MBEC)



Memory Protection

- Just before init, a singe VTL call sets the following immutable permissions:

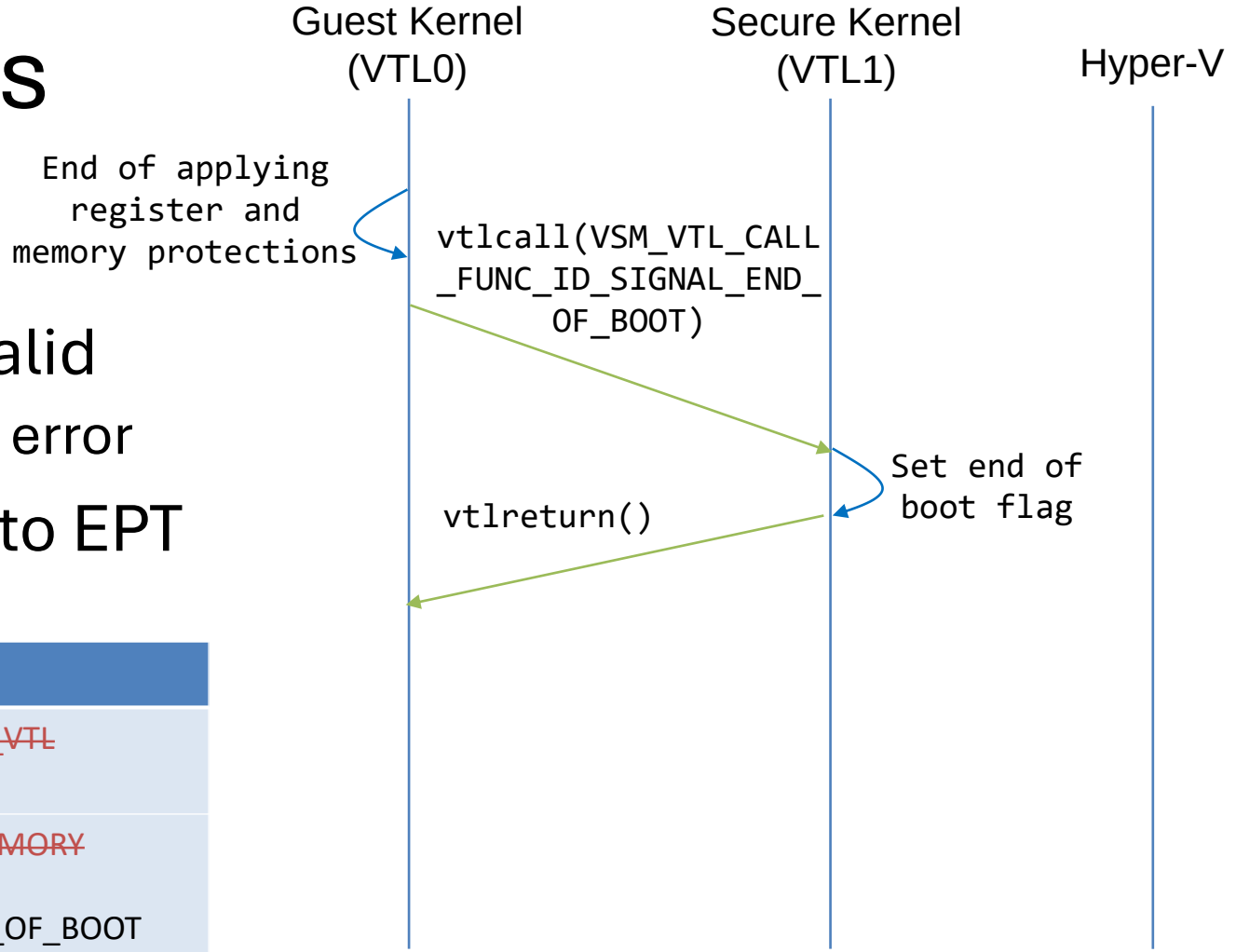
Region	Permissions
rodata	Read only
Text	Read, Kernel execute
VTL1 memory space	No access
Rest of the kernel memory	Read, Write, User execute

- Default permissions, without LVBS, were read, write, execute (user and kernel)

- If EPT access violation, memory intercept injected to VTL1, which raises a GP fault in VTL0

Lock down Protections

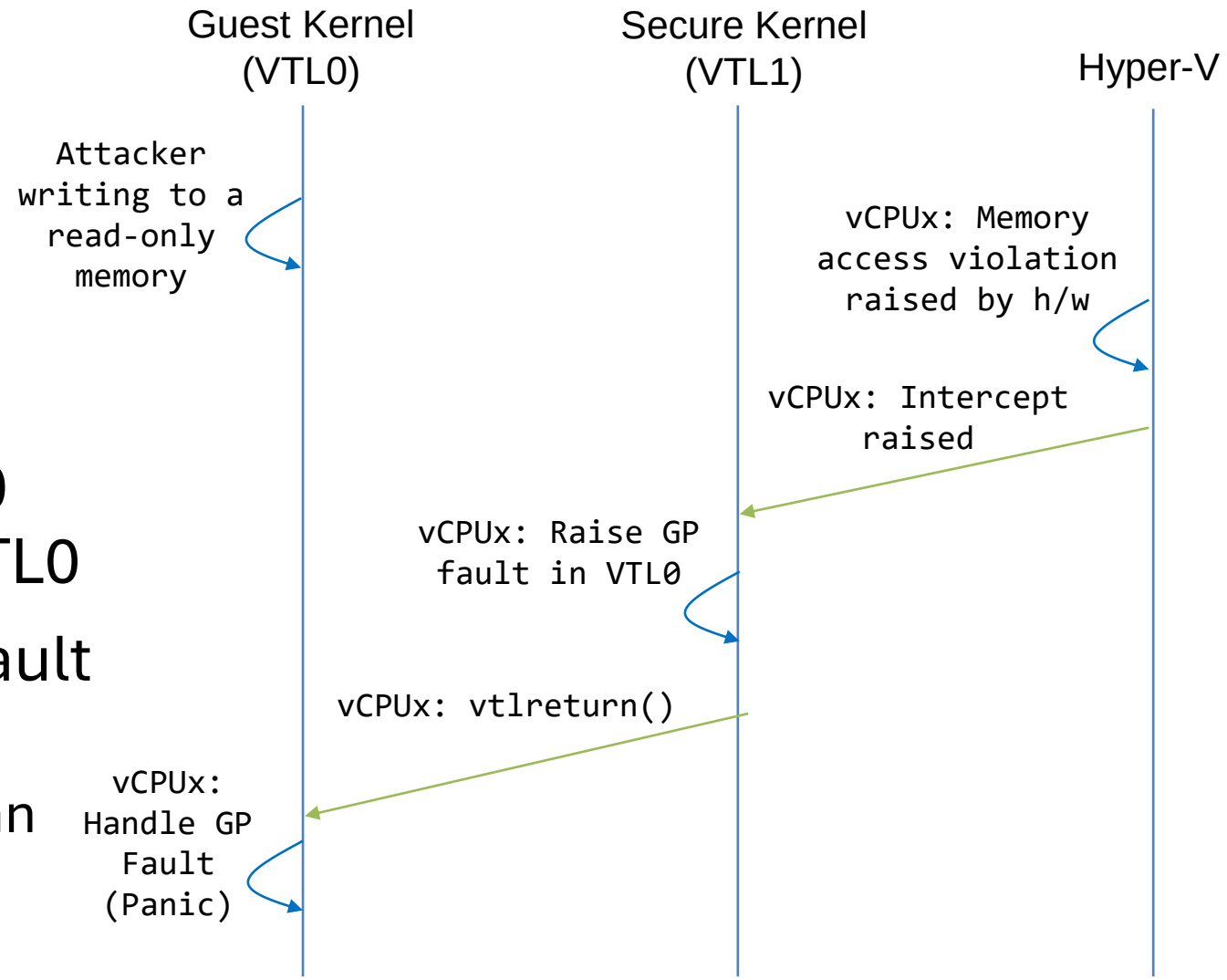
- Set end of boot flag in VTL1
- Certain vtlcalls are no longer valid
  - Attempts to invoke them returns error
- Beyond this point, all changes to EPT require authentication



Category	VTL Call Opcode
VTL1 Boot	~~VSM_VTL_CALL_FUNC_ID_ENABLE_APS_VTL~~ ~~VSM_VTL_CALL_FUNC_ID_BOOT_APS~~
Apply protections	~~VSM_VTL_CALL_FUNC_ID_PROTECT_MEMORY~~ ~~VSM_VTL_CALL_FUNC_ID_LOCK_CR~~ VSM_VTL_CALL_FUNC_ID_SIGNAL_END_OF_BOOT
Load VTL0 Data	~~VSM_VTL_CALL_FUNC_ID_LOAD_KDATA~~ VSM_VTL_CALL_FUNC_ID_COPY_SECONDARY_KEY
Module Loading/Unloading	VSM_VTL_CALL_FUNC_ID_VALIDATE_MODULE VSM_VTL_CALL_FUNC_ID_FREE_MODULE_INIT VSM_VTL_CALL_FUNC_ID_UNLOAD_MODULE

Exception handling

- Exceptions raised for
  - Violating memory access permissions
  - Violating register access
- VTL1 injects a GP fault in VTL0 and returns control back to VTL0
- VTL0 thread that caused GP fault is killed
  - Depending on configuration, can cause Kernel panic

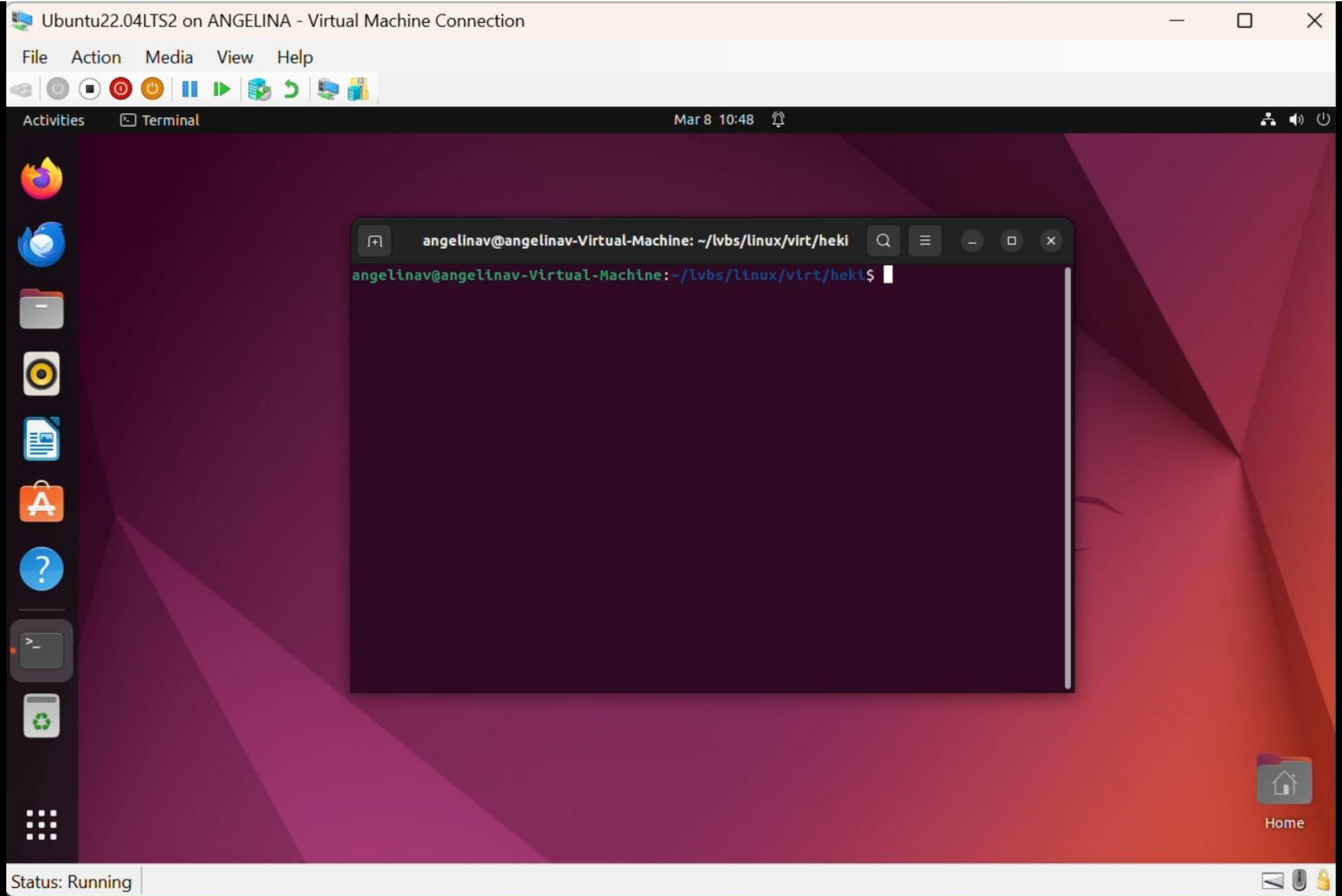


LVBS KUnit tests

- Optionally built-in tests for register locking and memory protections
- Tests are run after boot via debugfs

```
kunit_test_suites(
        &heki_x86_cr_disable_smep_suite,
        &heki_x86_cr_disable_wp_suite,
        &heki_x86_idtr_hack_suite,
#ifndef CONFIG_PARAVIRT_XXL
        &heki_x86_gdtr_hack_suite,
        &heki_x86_ldtr_hack_suite,
        &heki_x86_tr_hack_suite,
#endif
        &heki_x86_lstar_hack_suite,
        &heki_x86_star_hack_suite,
        &heki_x86_cstar_hack_suite,
        &heki_x86_efer_hack_suite,
        &heki_x86_apic_base_hack_suite,
        &heki_x86_sysenter_cs_hack_suite,
        &heki_x86_sysenter_eip_hack_suite,
        &heki_x86_sysenter_esp_hack_suite,
        &heki_x86_sfmask_hack_suite,
        &heki_x86_write_to_const_suite,
        &heki_x86_exec_suite
);
```

Demo: HEKI Tests

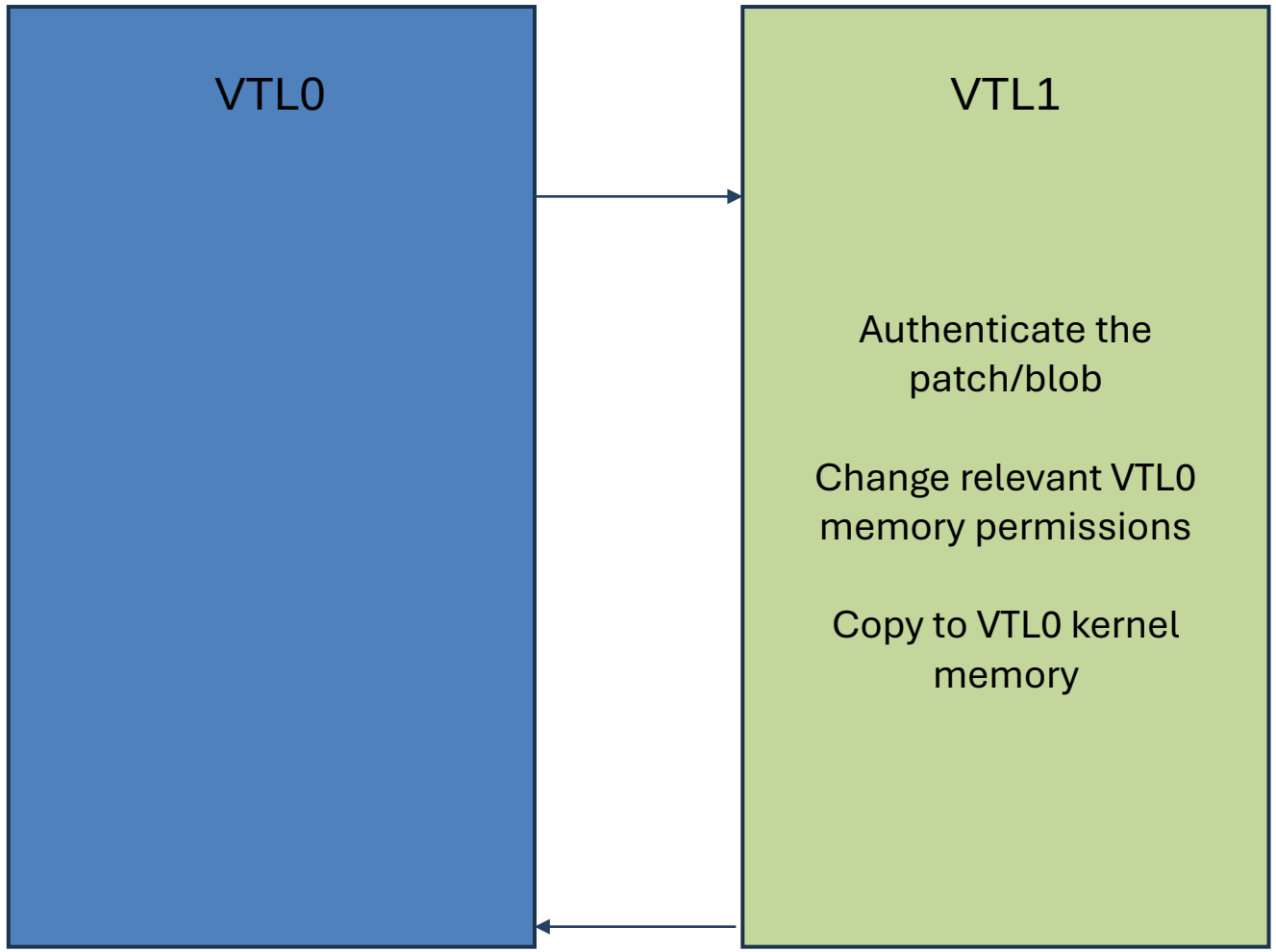

Ubuntu22.04LTS2 on ANGELINA - Virtual Machine Connection
File
Action
Media
View
Help
Activities
Terminal
Mar 8 10:48
angelinav@angelinav-Virtual-Machine: ~/lvbs/linux/virt/heki
angelinav@angelinav-Virtual-Machine:~/lvbs/linux/virt/heki$
Home
Status: Running

A look at kernel memory space:

Section	Permission enforced via LVBS	Text patching feature	Operation resulting in EPT violation
text	RX	Ftrace , Live patching, Jump Label Optimization, Static call optimization, Kprobes	RX -> RWX -> RX
rodata	R		
Data, .bss	RW		
module loading reserved memory	RW	Module loading	RW->R/RX
crash kernel reserved memory	RW	Kexec	RW->RWX
Rest of kernel memory space	RW	Ebpf-jit	RW->RX

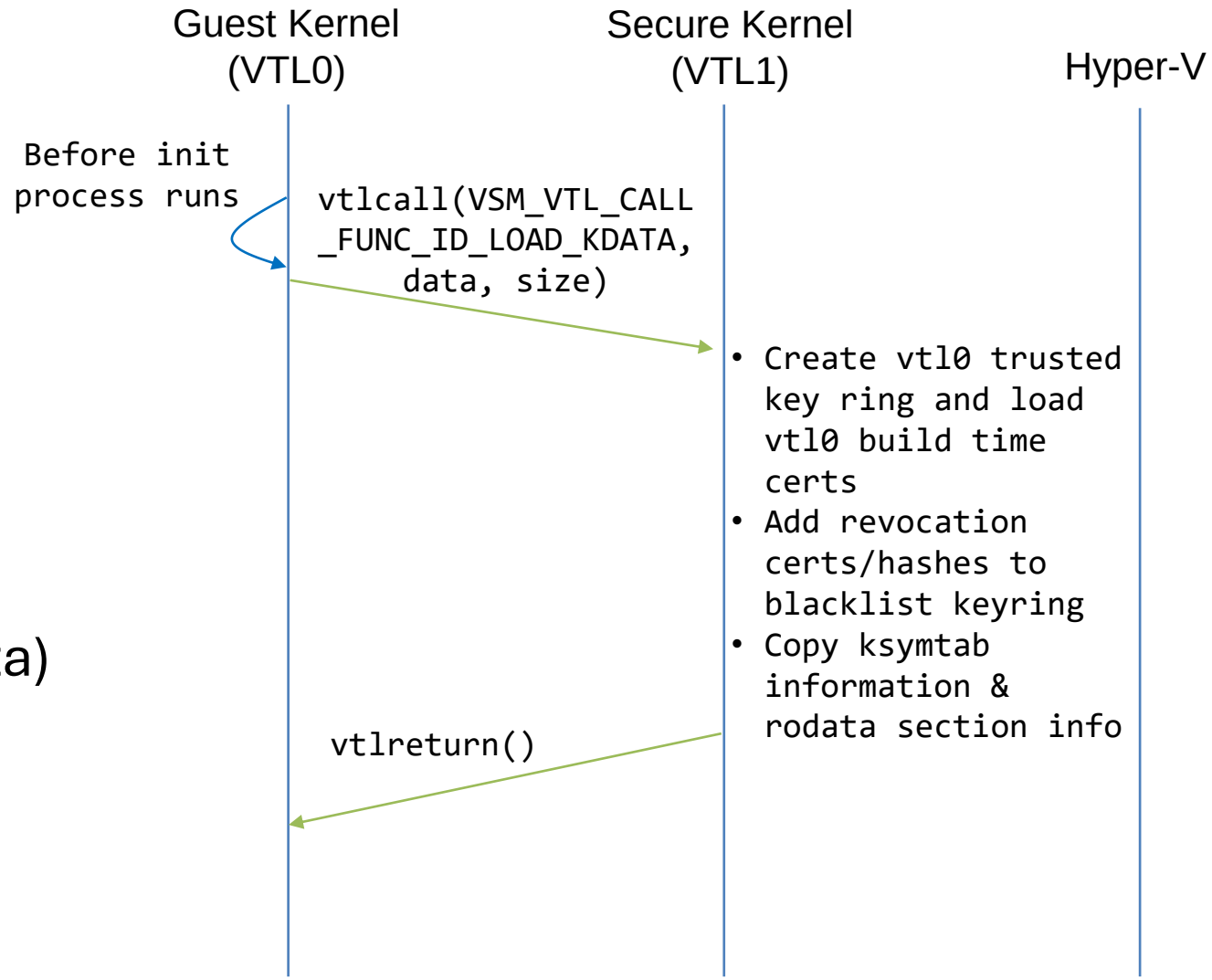
Text Patching Design Principle

- One atomic operation



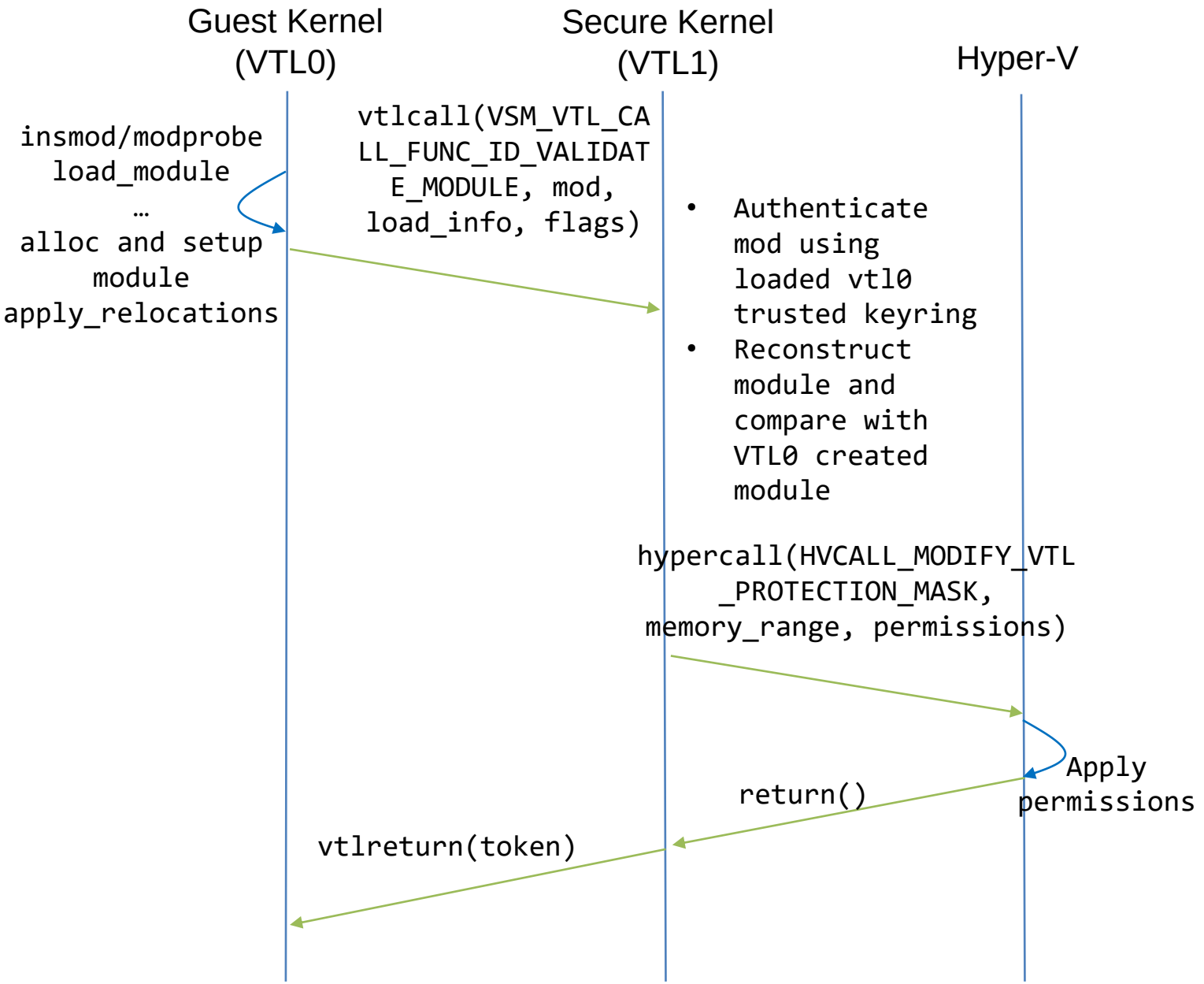
VTL0 Kernel Data

- Pass to VTL1
  - Build/Boot time certificates/keys
  - Blacklist/revocation certificates/keys
  - Kernel symbol table sections (ksymtab & ksymtab_gpl)
  - Read only data section (rodata)



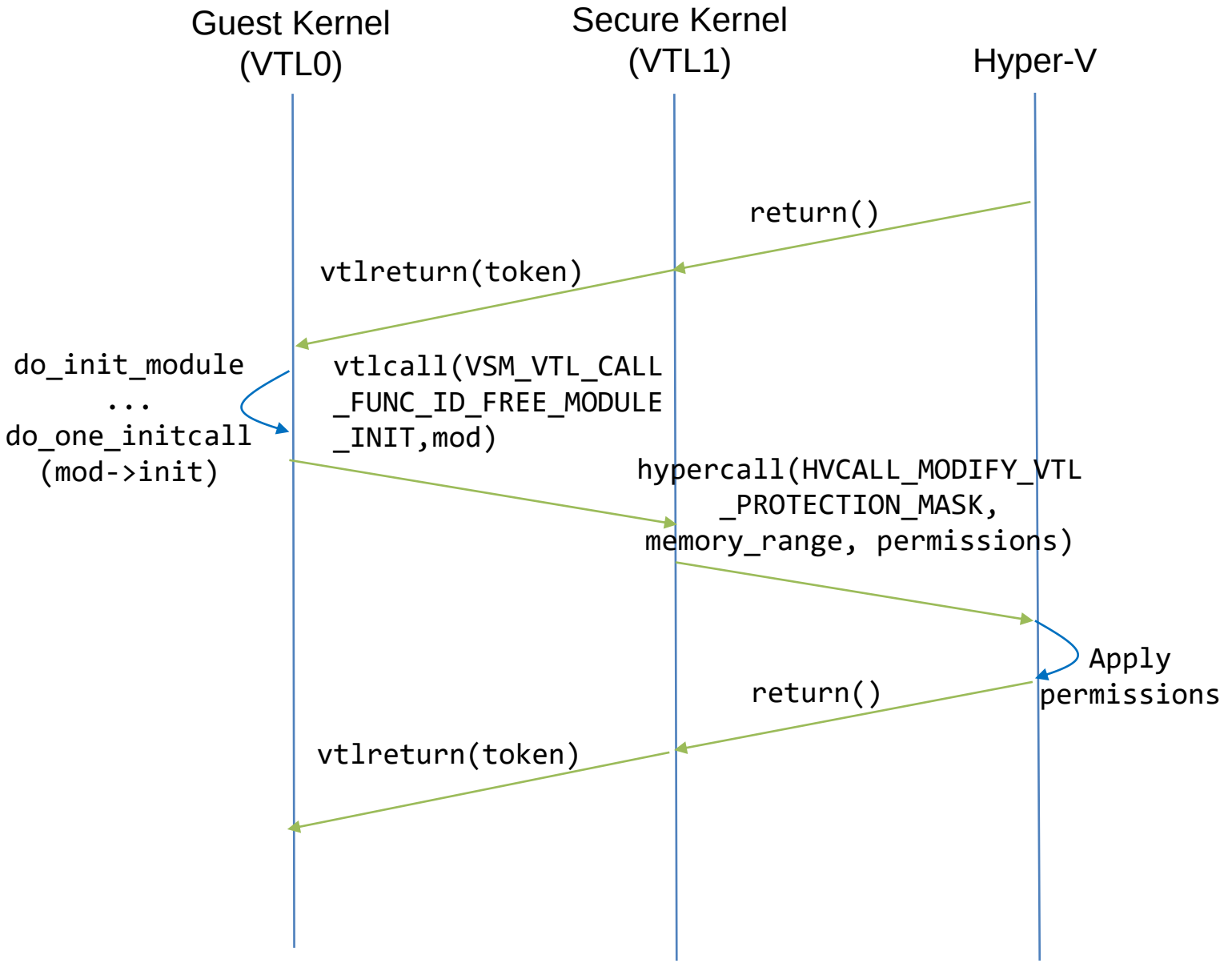
Module Loading

- VTL1 independently authenticates module
  - Verifies module signature
  - Reconstructs module and compares to VTL0
- VTL1 sends request to hypervisor to change EPT permissions for relevant module sections
- VTL1 returns per module secret token that can be used later



Module Loading

- After module init
  - Reset permissions of all module init sections to r-w
  - Set permission of read-only after init memory to ro



Module Loading

- Work in progress to support architecture dependent features:
  - Retpolines
  - Rethunks
  - Control flow integrity features

Need additional kernel data

Demo: Module Loading

\\.\pipe\Ubuntu22.04LTS2-com2relay - PuTTY

```
Auth memory matched for x_tables
Auth memory matched for ip_tables
Auth memory matched for efi_pstore
Auth memory matched for reed_solomon
Auth memory matched for ramoops
Auth memory matched for pstore_zone
Auth memory matched for i2c_core
Auth memory matched for parport
Auth memory matched for pstore_blk
Auth memory matched for lp
Auth memory matched for ppdev
Auth memory matched for parport_pc
Auth memory matched for msr
Auth memory matched for drm
Auth memory matched for vsock
Auth memory matched for hv_sock
Auth memory matched for sch_fq_codel
Auth memory matched for vmgenid
Auth memory matched for mac_hid
Auth memory matched for serio_raw
Auth memory matched for joydev
Auth memory matched for hyperv_fb
Auth memory matched for intel_cstate
heki: rapl: Auth memory mismatch for 2
mshv_vsm_validate_guest_module: Load guest mod
mshv_vsm_handle_entry: func id:0x1ffe6 failed
Auth memory matched for cryptd
Auth memory matched for crypto_simd
Auth memory matched for nls_iso8859_1
Auth memory matched for aesni_intel
Auth memory matched for sha1_ssse3
Auth memory matched for sha256_ssse3
Auth memory matched for ghash_clmulni_intel
Auth memory matched for drm_kms_helper
Auth memory matched for binfmt_misc
Auth memory matched for polyval_generic
Auth memory matched for polyval_clmulni
Auth memory matched for crct10dif_pclmul
Auth memory matched for drm_shmem_helper
Auth memory matched for intel_tcc_cooling
Auth memory matched for hyperv_drm
Auth memory matched for intel_rapl_common
Auth memory matched for intel_rapl_msr
```

Ubuntu22.04LTS2 on ANGELINA - Virtual Machine Connection

File Action Media View Help

Activities Terminal Mar 8 09:56

angelinav@angelinav-Virtual-Machine: ~/lvbs

```
angelinav@angelinav-Virtual-Machine:~/lvbs$
```

Home

Status: Running

What’s next

- Text patching features (Jump labels, static calls, ftrace, etc.)
- Kexec

Code

- VTL0 : https://github.com/heki-linux/lvbs-linux/tree/ubuntu-6.5-lvbs

- VTL1 : https://github.com/heki-linux/lvbs-linux/tree/secure-kernel-6.6-lvbs