Back To The Future: A Radical Insecure Design of KVM on ARM

Baibhav Singh
Rahul Kashyap

Introduction

- Samsung Research America
- Actively working on ARM system security

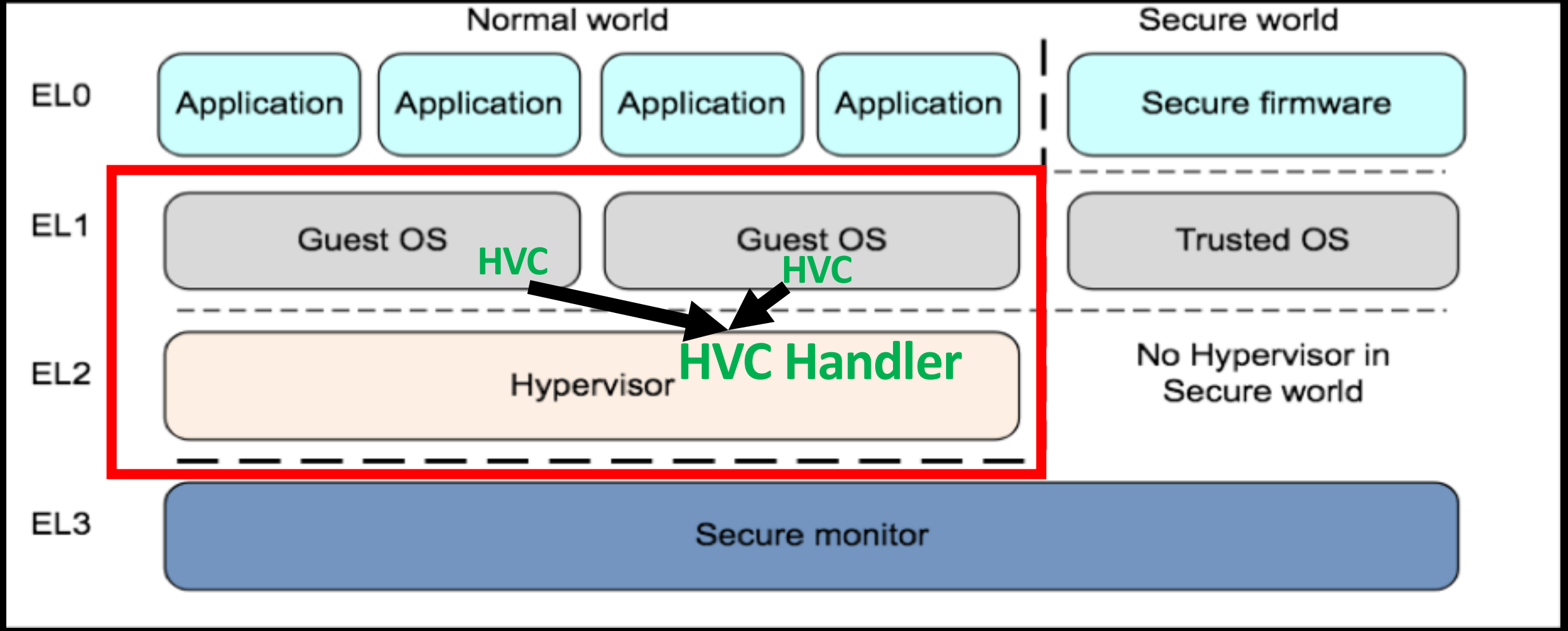
**The opinion present are my own and not necessarily represent the opinion of my employer*

Samsung Knox

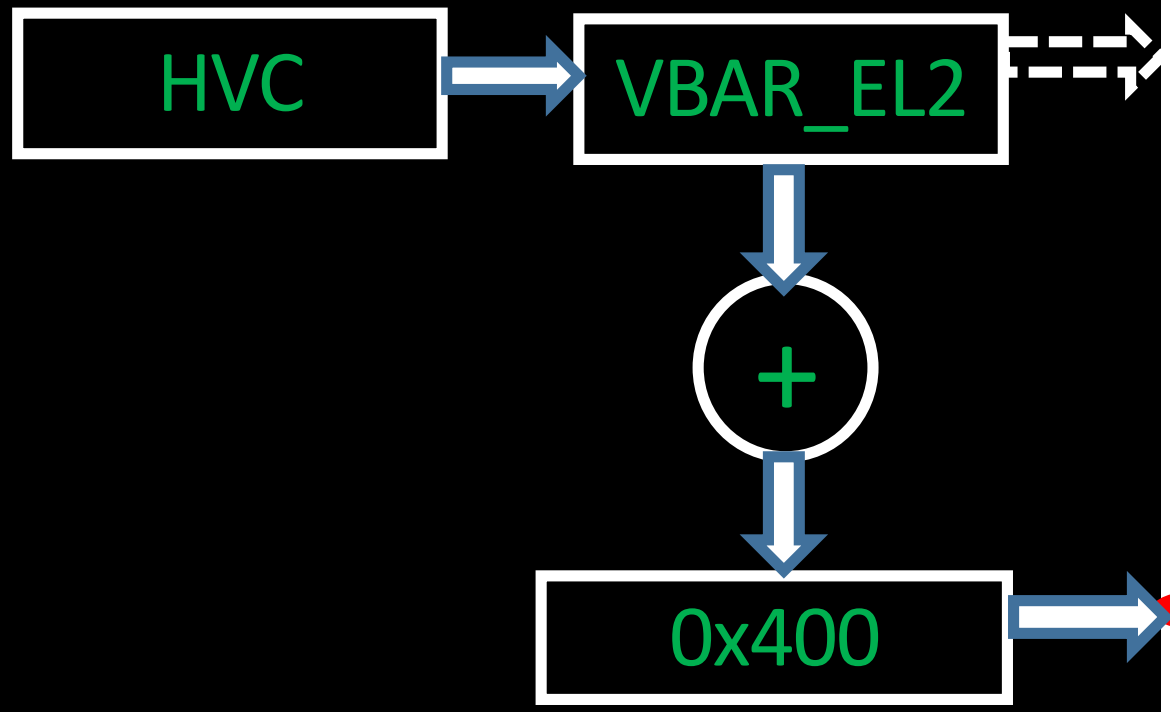
Background

- Found this issue while doing some hypervisors research
- On 25 Jan 2018, reported to Red Hat Product Security team Secalert@redhat.com provided detail report with exploit code
- Multiple mail exchange to help them understand the problem
- **Still is not fixed**
- Decided to submit a BH paper (Thanks to Rahul, Michael Grace)
- Thanks to BH for providing the platform

ARM v8-A Architecture



Exception Vector Table



Address	Exception type	Description
VBAR_ELn + 0x000	Synchronous	Current EL with SP0
+ 0x080	IRQ/vIRQ	
+ 0x100	FIQ/vFIQ	
+ 0x180	SError/vSError	
+ 0x200	Synchronous	Current EL with SPx
+ 0x280	IRQ/vIRQ	
+ 0x300	FIQ/vFIQ	
+ 0x380	SError/vSError	
+ 0x400	Synchronous	Lower EL using AArch6
+ 0x480	IRQ/vIRQ	
+ 0x500	FIQ/vFIQ	
+ 0x580	SError/vSError	
+ 0x600	Synchronous	Lower EL using AArch32
+ 0x680	IRQ/vIRQ	
+ 0x700	FIQ/vFIQ	
+ 0x780	SError/vSError	

ARM - Hypervisor

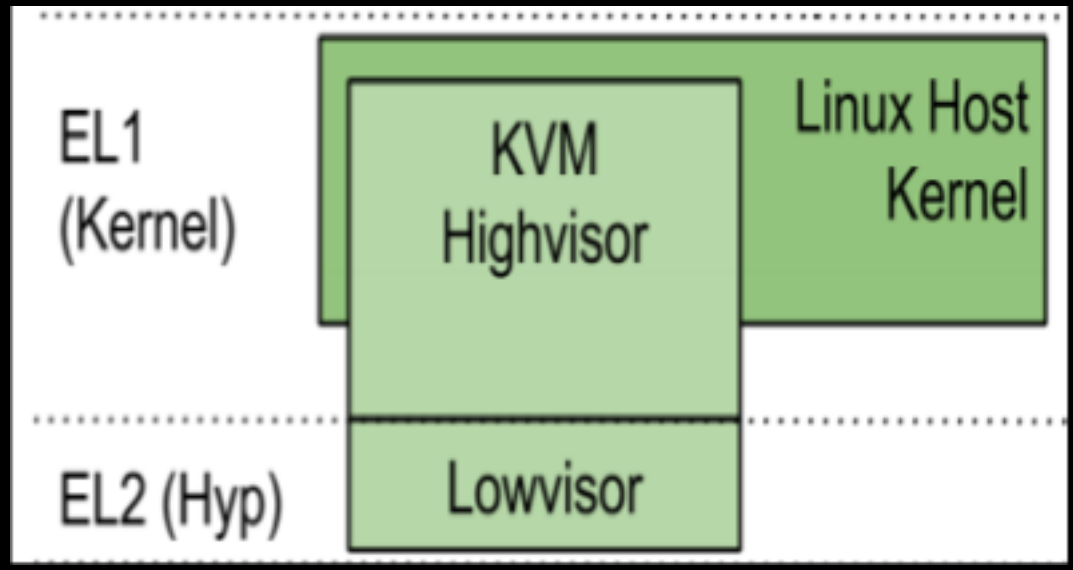
ARM Hypervisor executes in EL2

TYPE 1

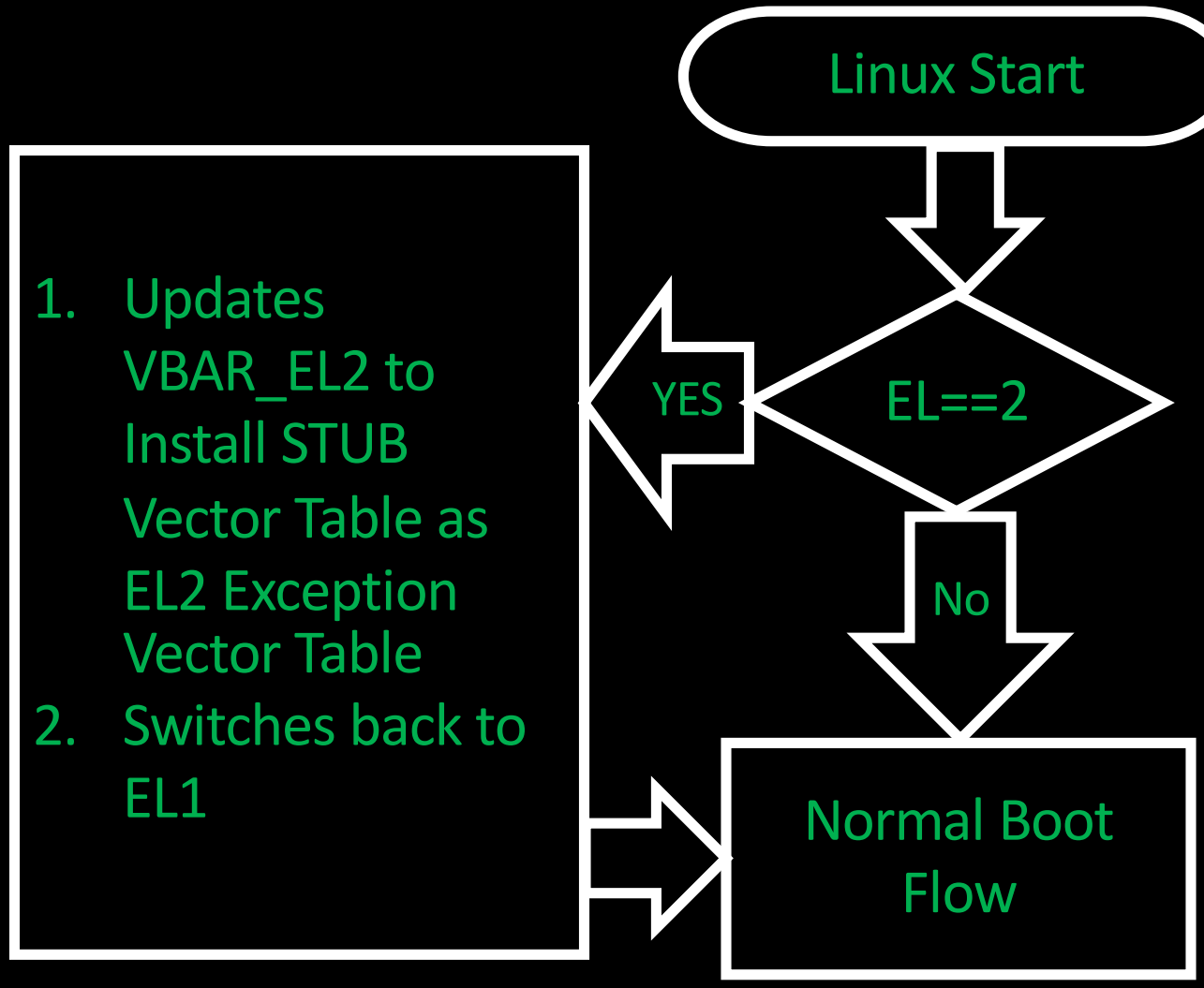
- Bare Metal Hypervisor
- Host is considered as VM
- XEN is type 1 and runs in EL2

TYPE 2

- Hypervisor is extension of Host Kernel
- Host is not considered as VM
- KVM is type 2 and runs partial in EL2



Linux and KVM




```
/*******************************************************************
 * Gets SPSR for BL33 entry
 *******************************************************************/
uint32_t rpi3_get_spsr_for_bl33_entry(void)
{
#.if RPI3_BL33_IN_AARCH32
    INFO("BL33 will boot in Non-secure AArch32 Hypervisor mode\n");
    return SPSR_MODE32(MODE32_hyp, SPSR_T_ARM, SPSR_E_LITTLE,
                DISABLE_ALL_EXCEPTIONS);
#else
    return SPSR_64(MODE_EL2, MODE_SP_ELX, DISABLE_ALL_EXCEPTIONS);
#endif
}
```

Stub Vector Table


```
ENTRY(  hyp stub vectors)
    ventry  el2 sync invalid          // Synchronous EL2t
    ventry  el2 irq invalid           // IRQ EL2t
    ventry  el2 fiq invalid           // FIQ EL2t
    ventry  el2 error invalid         // Error EL2t

    ventry  el2 sync invalid          // Synchronous EL2h
    ventry  el2 irq invalid           // IRQ EL2h
    ventry  el2 fiq invalid           // FIQ EL2h
    ventry  el2 error invalid         // Error EL2h

    ventry  el1 sync
    ventry  el1 irq invalid           // IRQ 64-bit EL1
    ventry  el1 fiq invalid           // FIQ 64-bit EL1
    ventry  el1 error invalid         // Error 64-bit EL1

    ventry  el1 sync invalid          // Synchronous 32-bit EL1
    ventry  el1 irq invalid           // IRQ 32-bit EL1
    ventry  el1 fiq invalid           // FIQ 32-bit EL1
    ventry  el1 error invalid         // Error 32-bit EL1
ENDPROC(  hyp stub vectors)
```

VBAR_EL2

Handler in case HVC originating from 64 bit Kernel

Filename : linux\arch\arm64\kernel\hyp-stub.s

Stub HVC Handler

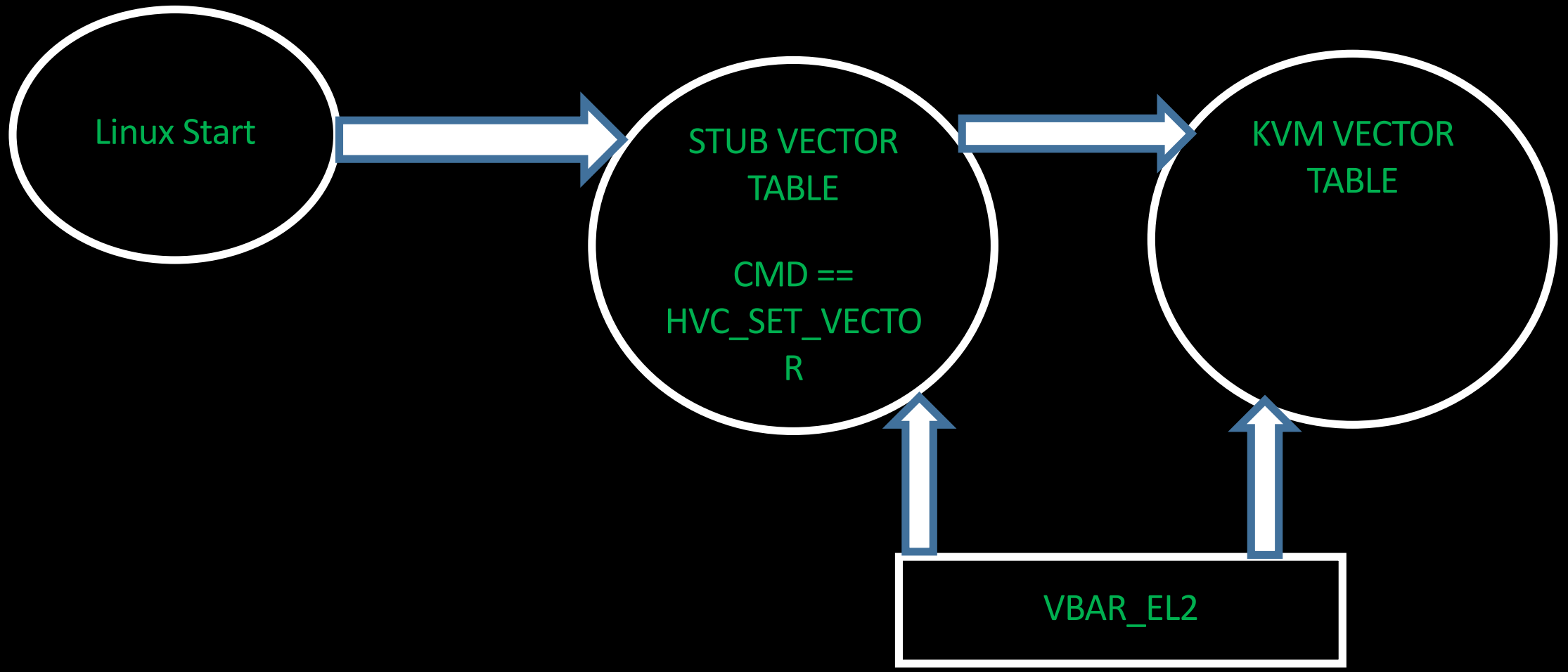

```
el1_sync:
    cmp x0, #HVC_SET_VECTORS
    b.ne    2f
    msr vbar_el2, x1
    b   9f

2:  cmp x0, #HVC_SOFT_RESTART
    b.ne    3f
    mov x0, x2
    mov x2, x4
    mov x4, x1
    mov x1, x3
    br  x4                  // no return

3:  cmp x0, #HVC_RESET_VECTORS
    beq 9f                  // Nothing to reset!

    /* Someone called kvm_call_hyp() against the hyp-stub... */
    ldr x0, =HVC_STUB_ERR
    eret

9:  mov x0, xzr
    eret
ENDPROC(el1_sync)
```

Checks if register X0 == 0

Updates VBAR_EL2

Filename : linux\arch\arm64\kernel\hyp-stub.s

State Diagram 1



KVM HVC Handler


```
el1_sync:                       // Guest trapped into EL2

    mrs x0, esr_el2
    lsr x0, x0, #ESR_ELx_EC_SHIFT
    cmp x0, #ESR_ELx_EC_HVC64
    ccmp    x0, #ESR_ELx_EC_HVC32, #4, ne
    b.ne    el1_trap

    mrs x1, vttbr_el2       // If vttbr is valid, the guest
    cbnz    x1, el1_hvc_guest   // called HVC

    /* Here, we're pretty sure the host called HVC. */
    ldp x0, x1, [sp], #16

    /* Check for a stub HVC call */
    cmp x0, #HVC_STUB_HCALL_NR
    b.hs    1f

    /*
     * Compute the idmap address of __kvm_handle_stub_hvc and
     * jump there. Since we use kimage_voffset, do not use the
     * HYP VA for __kvm_handle_stub_hvc, but the kernel VA instead
     * (by loading it from the constant pool).
     *
     * Preserve x0-x4, which may contain stub parameters.
     */
    ldr x5, =__kvm_handle_stub_hvc
    ldr_l   x6, kimage_voffset

    /* x5 = __pa(x5) */
    sub x5, x5, x6
    br  x5
```

Checks if HVC is from Host

Checks if register x0 < 3

Calls _kvm_handle_stub_hvc

Filename : linux\arch\arm64\kernel\hyp-entry.s

KVM HVC Handler

Filename :
linux\arch\arm64\kvm\hyp\hyp-entry.s

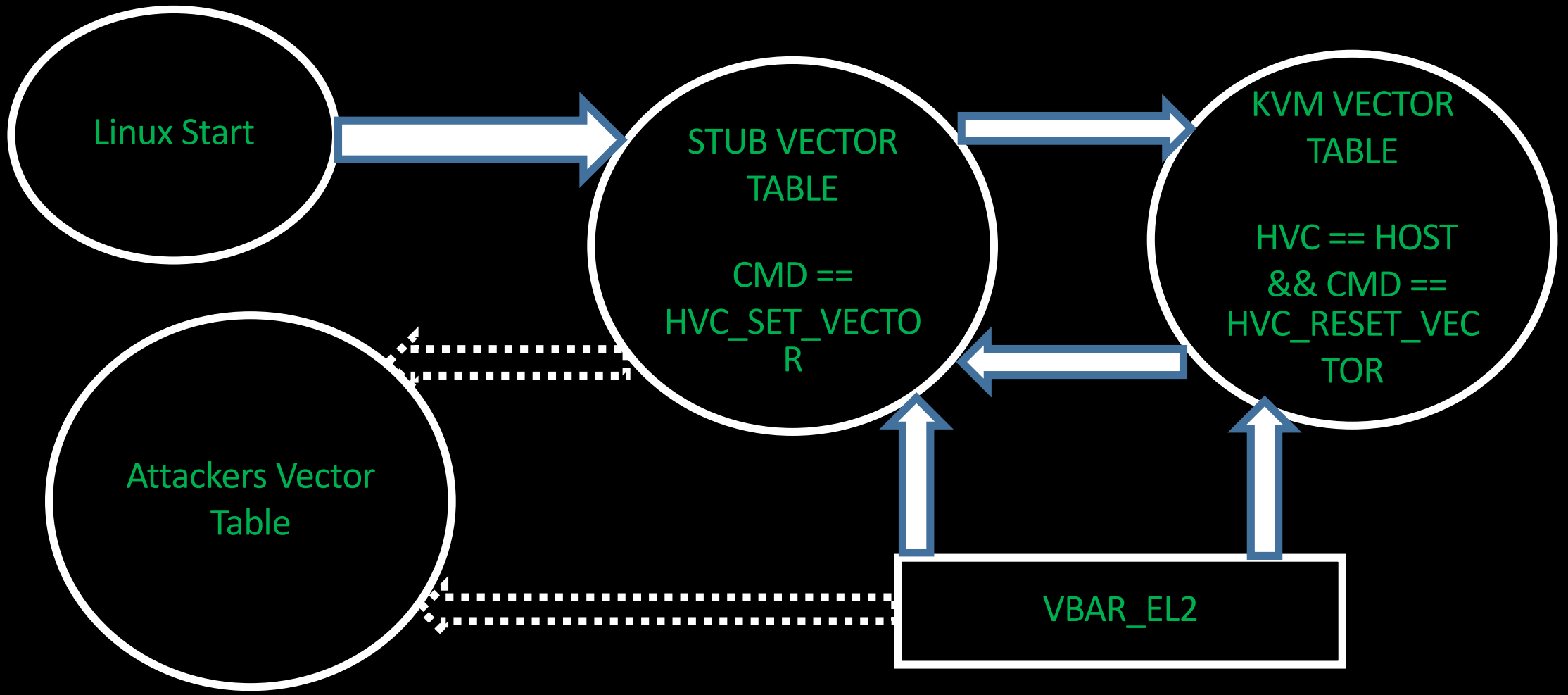
```
ENTRY(  kvm_handle_stub_hvc)
    cmp x0, #HVC_SOFT_RESTART
    b.ne    1f

    /* This is where we're about to jump, staying at EL2 */
    msr elr_el2, x1
    mov x0, #(PSR_F_BIT | PSR_I_BIT | PSR_A_BIT | PSR_D_BIT | PSR_MODE_EL2h)
    msr spsr_el2, x0

    /* Shuffle the arguments, and don't come back */
    mov x0, x2
    mov x1, x3
    mov x2, x4
    b   reset

1:  cmp x0, #HVC_RESET_VECTORS
    b.ne    1f
reset:
    /*
     * Reset kvm back to the hyp stub. Do not clobber x0-x4 in
     * case we coming via HVC_SOFT_RESTART.
     */
    mrs x5, sctlr_el2
    ldr x6, =SCTLR_ELx_FLAGS
    bic x5, x5, x6      // Clear SCTL_M and etc
    pre_disable_mmu_workaround
    msr sctlr_el2, x5
    isb

    /* Install stub vectors */
    adr_l   x5, __hyp_stub_vectors
    msr vbar_el2, x5
    mov x0, xzr
    eret
```

Checks register X0 == 2

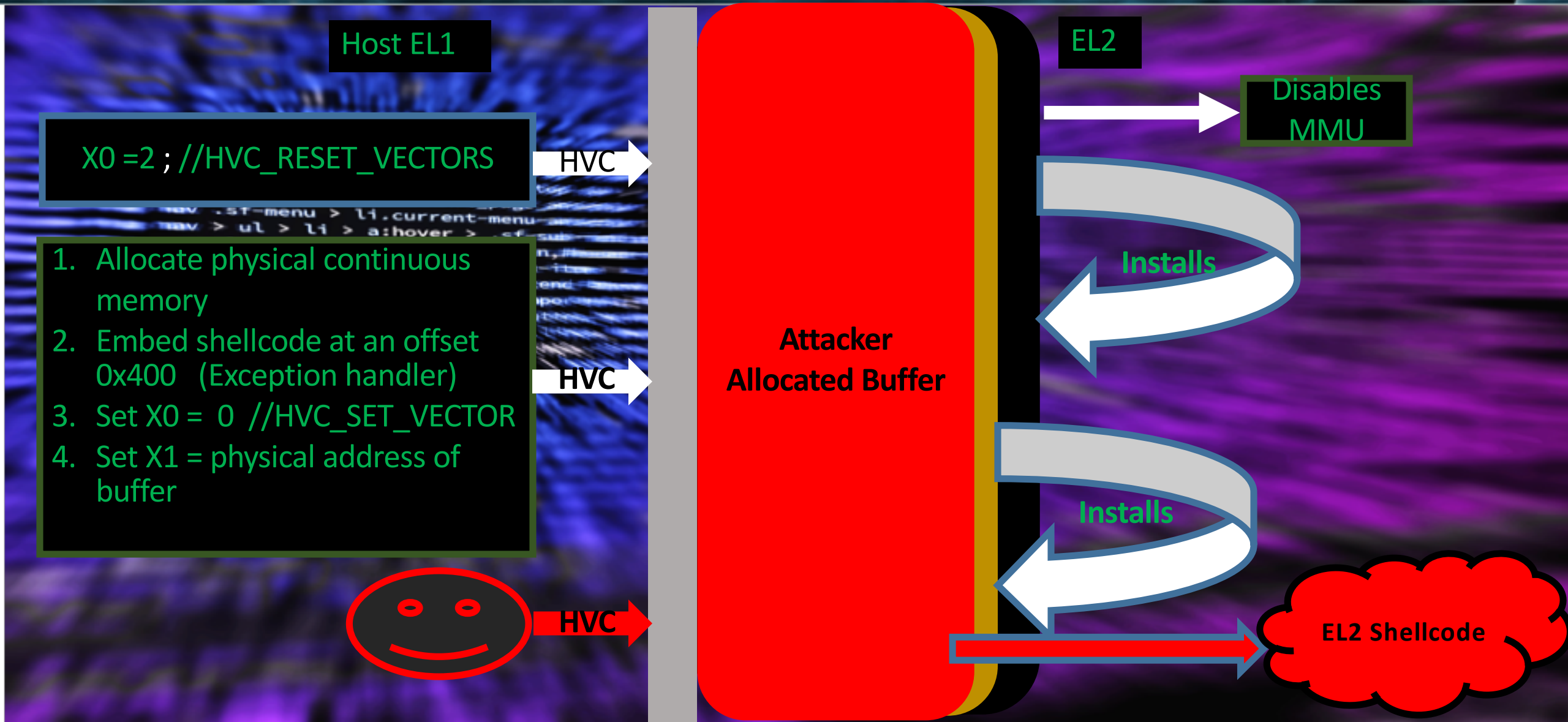
Disables EL2 MMU

Back to Stub Vector

State Diagram 2



Exploitation



KVM Team Response

"You're on the host, and you can break KVM by inserting a rogue kernel module. Big deal. You can also blast the page tables, corrupt file systems, and make sure the box is on fire"

KVM Threat Model Assumption : HOST.EL1 == EL2

Security hole in privilege isolation boundary

Host kernel compromise is End of the Game !

Real World : EL1 != EL2

For Attacker Beginning of a New Game.... ☺

Attacker can exploit this issue to gain more privilege and will migrate to EL2

- Launch attack from isolated and unreachable memory.
- Can configure EL2 to get code execution from various different places
- A generic way to bypass security implemented in the kernel (LKRG),by escaping to EL2
- Attack the secure monitoring running in hypervisor mode
- Gives attacker opportunity for Blue Pill for KVM on ARM

Juicy target for attacker to perform highly sophisticated and stealthy attack

Impact

Potentially bigger impact for mobile and IoT

- Most of them are ARM based
- Chances are high that it will boot in EL2
- Single Kernel Device (More Attack Surface)

Affected Architecture: ARM v7-A and ARM v8-A with hardware virtualization

Mitigation

My two cents… You're likely vulnerable to this attack.
Patch the system by making sure Linux starts in EL1

Reference

- https://static.docs.arm.com/ddi0487/ca/DDI0487C_a_armv8_arm.pdf
- https://developer.arm.com/products/architecture/a-profile/docs/100942/latest/hypervisor-software
- https://dl.acm.org/citation.cfm?id=2541946
- http://www.cs.columbia.edu/~cdall/pubs/atc17-dall.pdf
- http://www.cs.columbia.edu/~cdall/pubs/sosp2017-neve.pdf
- https://lwn.net/Articles/557132/
- http://lia.disi.unibo.it/Courses/som1516/materiale/VOSYS_BolognaKVMARM_2_12_2015.pdf