Stellux: Exploring Fine-Grain HW Privilege in a Clean-Slate Kernel

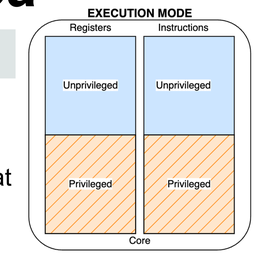
ComputerScience — Boston University

A single kernel binary designed for and executed at two hardware privilege levels.

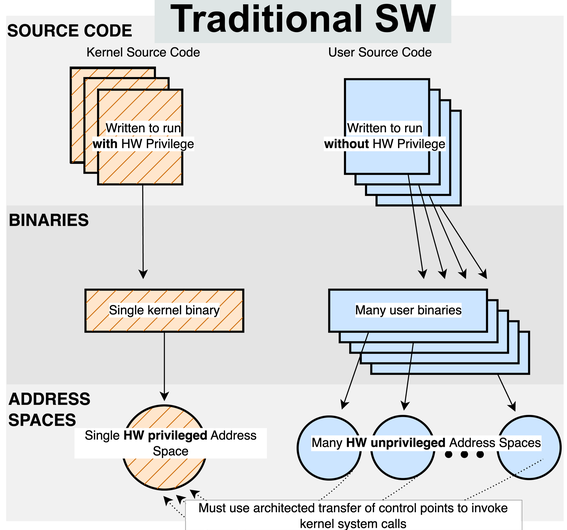
Idea

Traditional HW

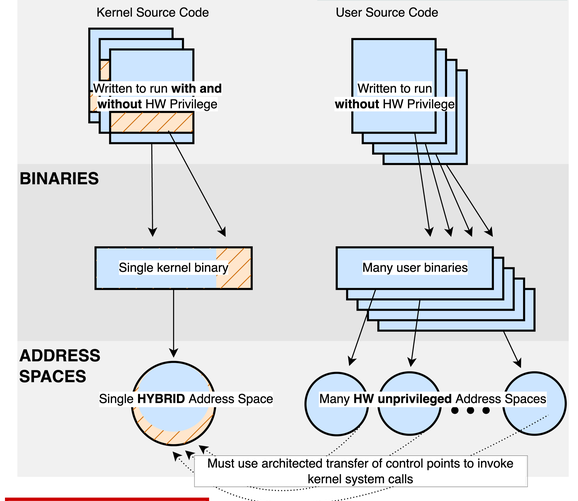
Offers two or more modes HW privileged execution. Can we design a kernel that exploits both?



Traditional SW



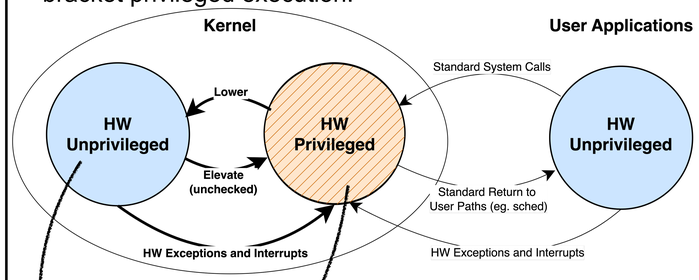
Stellux SW



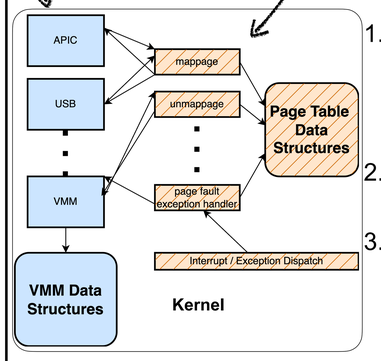
Design

New In-kernel Transitions

Triggered by two Primitives:1) Elevate and 2) Lower Kernel developers explicitly use Elevate() and Lower() to bracket privileged execution.

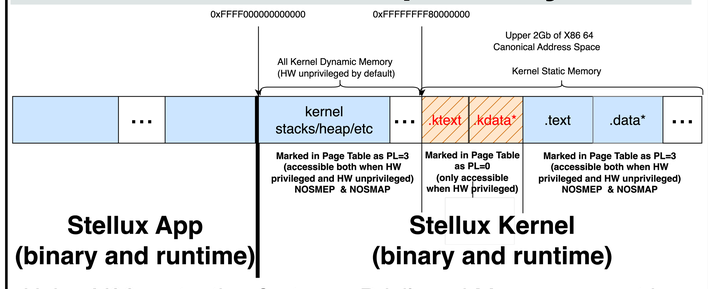


Stellux Hybrid Kernel Structure



1. A kernel composed of **mutually trusting** HW unprivileged and privileged code.
2. Explore minimizing HW-privileged code and data
3. Stellux is **NOT** a micro-kernel or a capability-based system.

Stellux Address Space Layout



Using X86 protection features, Privileged Memory cannot be accessed when executing without HW Privilege; however, both Privileged and Unprivileged Memory can be accessed when HW Privileged. Kernel dynamic memory is unprivileged.

Prototype

Elevate

```
void __kelevate() {
    __syscall(SYSCALL_SYS_ELEVATE, 0, 0, 0, 0, 0, 0);
    __set_elevated_usergs();
}
```

X86 **syscall** with **ret** to trigger ring transition — syscall handler does no checking other than isKernel()

Lower

```
void __klower() {
    __restore_lowered_usergs();
    __asm__ __volatile__ (
        "pushfq;"              // Push EFLAGS onto the
        "popq %%r11;"          // Pop EFLAGS into r11
        "cli;"                 // Disable interrupts
        "lea 1f(%%rip), %%rcx;"  // Load the address
        "movq %%gs:0x0, %%rax;" // Move the address
        "btrq $0, 0xf8(%%rax);"  // Set current->ele
        "sysretq;"              // Execute SYSRET and
        "1:"                   // Label for the next
```

Inlined lower carefully uses X86 **sysret** to switch to HW unprivileged execution with no flow control change

Static Privileged Data and Code

```
__PRIVILEGED_DATA
Framebuffer VGADriver::s_framebuffer;

__PRIVILEGED_DATA
Psf1Font* VGADriver::s_font = nullptr;

__PRIVILEGED_DATA
uint32_t* s_backBuffer;
```

```
__PRIVILEGED_CODE
void loadIdtr() {
    // Load the IDT
    asm volatile ("lidt %0" :: "m"(g_kernelIdtDescriptor));
}
```

Simple decorations used to mark static code and data for placement in “privileged” memory

Hybrid Kernel Code

```
Apic::Apic(uint64_t base, uint8_t spuriousIrq) {
    if (!g_lapicPhysicalBase) {
        g_lapicPhysicalBase = (void*)base;
        // Map the LAPIC base into the kernel's address space
        g_lapicVirtualBase = (volatile uint32_t*)zallocPage();
        RUN_ELEVATED({
            paging::mapPage(
                (void*)g_lapicVirtualBase,
                g_lapicPhysicalBase,
                USERSPACE_PAGE,
                PAGE_ATTRIB_CACHE_DISABLED,
                paging::g_kernelRootPageTable,
                paging::getGlobalPageFrameAllocator()
            );
            paging::flushTlbPage((void*)g_lapicVirtualBase);
        });
    }
    // Set the spurious interrupt vector
    uint32_t spuriousVector = read(0xF0);
    spuriousVector |= (1 << 8);     // Enable the APIC
    spuriousVector |= spuriousIrq; // Set the spurious interrupt
    write(0xF0, spuriousVector);
}
```

Mutually Trust:
HW unprivileged code operates knowing that it can “directly” invoke HW privileged code.

Bracketed HW Privilege
Elevate and lower to invoke hw privilege code and manipulate hw privileged data. HW privileged code operates knowing it was invoked by the trusted kernel HW unprivileged code.

BOSTON UNIVERSITY
Author & Developer: Albert Slepak Undergraduate
Advisors: Thomas Unger Ph.D & Professor Jonathan Appavoo

With support from:
Red Hat Research