Supervisor Memory Protection

Supervisor Memory Access Protection (SMAP) and Supervisor Memory Execute Protection (SMEP) are two kernel security mechanisms present in Broadwell and later Intel CPUs. SMEP prevents the kernel running in ring 0 from executing code which is user accessible. SMAP prevents the kernel from accessing userspace memory while the AC flag in the RFLAGS register is clear. These features can help harden the kernel against exploitation and prevent certain kinds of memory corruption. SMEP and SMAP are described in Volume 3A, 4.1.1 of the Intel Manual.

Checking for Feature Availability

The CPUID instruction can be used to check if SMEP and SMAP are available. CPUID will report whether SMEP and SMAP are reported in leaf 7, bits 7 and 20 of the EBX register respectively.

Initializing SMEP and SMAP

SMEP and SMAP can be initialized by setting bits 20 and 21 of CR4 respectively.

```
#define CPUID_SMEP 7
#define CPUID_SMAP 20
#define CPU_CR4_SMEP_BIT 20

#define CPU_CR4_SMAP_BIT 21

void supervisor_memory_protection_init(void) {
    uint32_t eax, ebx, ecx, edx;
    eax = 7;
```

```
    ecx = 0;
    cpuid(&eax, &ebx, &ecx, &edx);
    if (ebx & CPUID_SMEP) {
        cpu_cr4_set_bit(CPU_CR4_SMEP_BIT);
        log(Log_Info, "SMEP Enabled");
    }

    if (ebx & CPUID_SMAP) {
        cpu_cr4_set_bit(CPU_CR4_SMAP_BIT);
        log(Log_Info, "SMAP Enabled");
    }
}
```

Reading and Writing User Memory with SMAP Enabled

Sometimes it may be necessary to modify user space accessible memory from ring 0, such as when setting up the stack for user space code, or performing certain syscalls. In these cases, SMAP protections can be disabled by setting the AC flag. This flag can be set and cleared with the STAC and CLAC instructions. A version of memcpy which allows copying data from ring 3 to ring 0, or vice versa, may look like this:

```
static inline void cpu_flags_set_ac(void) {
    // Set AC bit in RFLAGS register.
    __asm__ volatile ("stac" ::: "cc");
}

static inline void cpu_flags_clear_ac(void) {
    // Clear AC bit in RFLAGS register.
    __asm__ volatile ("clac" ::: "cc");
}
```

```
void *user_memcpy(void *destination, const void *source, si
    // Disable SMAP protections.
    cpu_flags_set_ac();
    // Perform normal memcpy.
    void *ret = memcpy(destination, source, size);
    // Restore SMAP protections.
    cpu_flags_clear_ac();
    return ret;
}
```