HW2

Haoran Lei

Q1

The kernel stacks are protected and separated from the user stacks. It is in a different physical location, and not accessible by instructions made by the OS in user mode. The reason of separation is for separation of privileges and security.

User processes can modify their user stacks in an arbitrary invalid way, therefore the kernel cannot trust the userspace stackpointer to be valid nor usable. However, the kernel stacks are set to be under control of the privileged instructions in kernel mode and thus much safer to operate on, since invalid memory requests or instructions will be ruled out before the kernel allows a process to execute on a kernel stack.

Example: In Lab 1 hijacking question, without a switch from user stack to kernel stack when a process makes systemcalls, the kernel can operate on an already messed-up user stack (also kernel stack, since XINU does not separate user/kernel stacks) with return-address modified and pointing to a location in text segment which should have never be accessed. The calling of a systemcall sleepms() should have protected the stack of the caller until it returns from a system call if a switching stack technique had been applied. Instead, in our hijacking, the return address of the caller is easily modified by another user process due to the transparency of the system's memories to the user process. The result is the breakdown of the kernel.

Q2

Virtualization employs techniques used to create instances of an environment, as opposed to simulation, which models the environment; or emulation, which replicates the target environment such as certain kind of virtual machine environment. Full virtualization requires that every salient feature of the hardware be reflected into one of several virtual machines – including the full instruction set, input/output operations, interrupts, memory access, and whatever other elements are used by the software that runs on the bare machine, and that is intended to run in a virtual machine. In such an environment, any software capable of execution on the raw hardware can be run in the virtual machine and, in particular, any operating systems. The obvious test of full virtualization is whether an operating system intended for stand-alone use can successfully run inside a virtual machine. (source: Wikipedia)

Overhead best case: privilege/sensitive instructions are not executed at all, all instructions of the guest OS is executed natively.

Overhead worst case: privileged instruction often used, and guest OS often traps to the hypervisor and context switch happens a lot—large overhead.

The x86 architecture has sensitive non-privileged instructions which, as described by Popek and Goldberg, can change the underlying resources (e.g. doing I/O or changing the page tables) or observe information that indicates the current privilege level (thus exposing the fact that the guest OS is not running on the bare hardware). Threrefore, there are 17 instructions that silently behave differently in privileged versus non-privileged mode.

popf: in user mode, popf instruction pops flags saved on stacks, but privileged flags (i.e interrupt-enable flag) will be ignored; in kernel mode, popf (which is invoked by kernel code) can now pop privileged flags. To be virtualizable, popf should cause traps in user mode (or guest OS) so that the hypervisor can detect when guest OS wants to change its interrupt level.