CS261: Notes on Memory Protection

Thibaud Hottelier

September 8, 2009

1 Multi-users Machines

The first security models were designed with multi-users machines in mind. The goal was to prevent users of the same machine from interfering with each other. The main objective was therefore isolation. Here are three possible ways to achieve it:

- Use two machines. This trivially fulfills the goal but it is not cost efficient.
- Use one machine with an operator that takes care running and shutting down each job. The operator, who is trusted, has to reset all states between each job. This includes persistent storage which must be erased. This design is analogous to virtual machines and the special sessions used by webcafés.
- Use one machine with an extra layer of indirection when addressing memory. User-space processes deal only with virtual-addresses with are translated into real (a.k.a. physical) addresses by the MMU (Memory Management Unit) for each access. The MMU is programmable only by the kernel, which will take care of setting up the MMU translation table such that each process can only access its own memory. The kernel updates the MMU tables after each context-switch. Here, the O.S. is trusted. This design requires the CPU to be able to distinguish the privileged programs (the kernel) from the unprivileged ones (the user-space processes). On the x86 architecture, this is done with ring levels. User-space processes can only enter in the privileged mode through interrupts or traps (i.e. system calls). This design is vulnerable to D.O.S. attacks as well as side channel attacks. For instance, one process can guess how much time another process is waiting because of cache misses. By playing with the CPU caches, this information can then be used to derive the page access pattern.

2 Isolation with controlled Interactions

One simple design to ensure isolation between processes while still allowing communications would be to isolate each process in a container. Containers can communicate through messages that are passed along by the kernel. This models micro-kernels for instance. For efficiency reasons, messages are implemented through shared memory (i.e. zero-copy). Once a page is shared between multiple processes, it must be mapped as a read only page for both the sender and the receivers. This prevents:

1. the sender from modifying the message while the receivers read them.

2. the receivers from changing the message before the shared page is returned exclusively to the sender.

Note that with the zero-copy mechanism, the accounting of memory usage becomes difficult. Who do you charge you for a message: the sender or the receivers? This opens this design to D.O.S. attacks.