More on Side Channels: Timing, Termination and Memory traces

April 3rd, 2014

Timing- and Termination-Sensitive Secure Information Flow: Exploring a New Approach

Discussion about figure 2
Program (a) will be detected in weak security model to have information flow because different branches has different number of statements. However, it will not detect the information flow in second program, where there are same number of statement. However, in the second program, if we take caching behavior into account, we will realize that the two branches take different amount of time to run. The strong security model will capture this.

Discussion about figure 3
Similar to figure 2, the different also exists in termination channel. Weak termination sensitive model will only detect program(a) as one branch will never terminate. However, program(b) can only be detected in strong termination sensitive model.

Main idea
The main idea is to partition the program into pieces and run them concurrently so that no information is leaked. The way to enforce it is to use a scheduler with different policies. There are three guarantee: insensitive, weak-insensitive, strong-insensitive.

Bugs in SME paper
SME claims that their approach ensures weak-sensitive non-interference for all security levels, which is not correct for non-comparable labels. In order to make the claim correct, we need to transform partial ordered security lattice into a total ordered one. In detail, for a partial ordered lattice, we created a total
order for it only for the purpose of scheduling, and then just run the lowest thread that is ready.

**Memory-trace oblivious program execution**

**Background**

We consider the situation where a TPM is employed and CPU is temper resistant. All the hardware is malicious except CPU. Here, only encrypting memory is not enough as memory access trace can still leak information. Cryptographic tool call ORAM can obfuscate memory traces but can incur a big overhead.

**Main idea**

The paper uses programming language techniques to offer efficient memory-trace oblivious program execution, while providing formal security guarantees. It also provides several ways of optimization to reduce the overhead incurred by ORAM, including multi-ORAM bank and using public arrays in certain cases.

**memory trace obliviousness**

Memory trace obliviousness is another kind of non-interference, in that it ensures that no matter what sensitive input it is, it will always produce the same indistinguishable memory traces.