CS 5341

Oscar Galindo

**Spectre Attacks: Exploiting Speculative Execution**

This paper begins by focusing on the steal-information-schemes that were successful in the past. Previously, other schemes have successfully used “power consumption, electromagnetic radiation, or acoustic noise to extract cryptographic keys as well as other secrets.” The paper continues to point out that this time around a scheme is proposed that exploits the architectural performance enhancement called speculative execution.

Under speculative execution the computer, basically, makes predictions of what code should be executed when branching code is encountered during execution. This means the computer decides among some branches and predicts which branch will likely run based on previous experiences and the code in the predicted branch is ran. If the prediction is wrong after the needed values for a decision are known the state of the machine is reversed to run the correct branch, if the correct branch was run ‘speculatively’ then the computer sees a performance boost and the code can continue. To explode this ‘speculative execution’ mechanism the paper proposes two possible schemes.

The first scheme relies on ‘mis training’ the branch predictor so that a branch of the branching code is assumed to be ran always. Then, to attack, a value or values that must be used in the branching condition-evaluation is or are changed to something malicious and used during speculative execution as a dependency. The reasons why speculative execution may occur are varied (e.g., cache misses, congestion of an execution unit, or complex arithmetic dependencies) nevertheless the code in the predicted branch runs even if the dependencies are (i.e., needed value(s)) changed to a potentially malicious value. While the execution state of the computer is normally undone after the lines are speculatively executed, the results/values accessed by running those lines will be placed in cache. From cache the attacker can detect through Flush+Reload or Prime+Probe what memory location is brought into cache, this in turn reveals the value of data that was maliciously obtained through speculative execution. Finally, the gathered information from the Spectre attack is transmitted to the attacker via a covert channel (e.g., a socket).

The second scheme relies on ‘poisoning’ indirect branches or the manipulation of predicted addresses for branches. As suggested by the name, the idea behind this variant is to train the branch predictor to run code in at an address where a set of instructions is stored and targeted with malicious purposes. In addition, the scheme needs the location of a, so-called, ‘spectre gadget’ which is essentially code-fragments already in the user’s memory space that allow the attacker to leak information to the covert channel (e.g., a socket). In summary, the scheme proposes to mis train the branch predictor to run a ‘spectre gadget’ to leak information. While defenses like ASLR (Automatic Space Layout Randomization) exist to block randomize the address location of instructions in the user’s memory space, the paper shows it is possible to circumvent this protection, as well as hypervisors, and different configurations of BHBs (Branch History Buffers).

This work is overall frustrating if one feels that one must work to prevent this attack. The attack seems to be almost unpreventable, especially because what the paper offers as a conclusion “As a result, we believe that long-term solutions will require fundamentally changing instruction set architectures. Among all the defense mechanisms that the paper refers to the most feasible one in my opinion is preventing data from entering covert channels, there seems to be an opportunity to create a mechanism that acts as a hypervisor, or MMU (Memory Management Unit) whose only task is to prevent the scenario where data is accessed to be sent through covert channels. Overall, I think spectre attacks are harmful and should be strived-to-be removed from the system.