CMP SCI 635: Modern Computer Architecture

Spectre Attacks: Exploiting Speculative Execution

Name: Kunjal Panchal Date: 7th Nov, 2019 Student ID: 32126469

Paper: 17 – Security [Kocher17]

Strengths:

1. When the exploit does work, it takes forever to scan memory and find anything useful. From multiple experiments described on internet, on multiple machine and multiple configurations; it would take about a month to scan 4GB of memory, and any target information would've gone in and out of memory in that time frame. Without countermeasures reading memory using Spectre was already very slow, 1500 bytes per second on high end machines, amplification makes that even slower.

2. Spectre doesn't work in virtual machines, as the lag between the physical hardware and the abstraction layer is too great.

Weaknesses:

1. If the attack is on the business of dealing with secure data (credit cards, financial data, social security, certain levels of PII, including medical data), it absolutely should be running on bare metal hardware and only allowing trusted processes. Any system admin who permits otherwise in such an enterprise scenario is guilty of incompetence. So technically, server and cloud enterprises shouldn't be vulnerable to it.

2. Spectre is a vulnerability, but one that cannot be easily exploited. One has to use a newer CPU on an older operating system, one that doesn't randomize memory addresses, a lot of the "patches", one hears about are just strengthening pre-existing mitigations to such attacks, or disabling the performance enhancements entirely.

Questions/Assertions:

1. VLIW-family CPUs like Microsoft's EPIC experiment could eliminate the problem by doing speculative execution in the compiler. Compiler-based speculation has some advantages, notably that the only information it leaks is information from the system it was compiled on. The actual hardware that it runs on doesn't leak because it isn't speculating anymore.

2. Is it possible to flood the side channels with random gibberish to hide the important data?

3. Put the data in a small bit of restricted memory that cannot be accessed outside of speculative operations. If the speculative operations have been decided to be the instructions that should be, move the state to regular memory.

4. Implementing site-isolation, which entails basically rebuilding the entire Javascript engine of every modern browser to include strict timing and cache barriers on basically every read and write. This means taking the Javascript engine from a normally written piece of code and converting it into the writing style employed by multithreaded applications, but everywhere instead of just where threads are sharing information and then not taking any advantage of that rewrite by keeping the process on a single, isolated thread.

5. Can we make a software which will be contained in fully isolated VMs?