I. Sennovskiy

isennovskiy@gmail.com

A new post-meltdown technique. Using speculative instructions for checking virtualization.

1. Intro

Meltdown attack uses speculative execution for accessing contents of memory, that an unprivileged attacker would not be able to view. This is possible because CPU executes certain instructions preemptively to speed up execution. Meltdown uses accesses to attacker-control buffers in speculative execution, changing the state of CPU cache in such a way, that the attacker can use memory access timing as a side channel.

However, speculative execution can be used, as we'll show, to check if certain instructions are deemed "light" by CPU.

2. Virtualization

VT-x technology in Intel CPUS allows hypervisor to choose for certain instructions, whether a VMEXIT will happen, for example ***rdtsc***. Most virtualized environments in standard configuration are set to create a VMEXIT on ***rdtsc***, including but not limited to Virtualbox, VMware, Parallels running on Apple hypervisor and Parallels hypervisor. VMEXIT in itself is a change of context of execution, which will be useful later.

3. The attack

A buffer is created, that spans several pages. The attack differs from Meltdown, it doesn't use a cache timing threshold. Instead of speculatively accessing regions of memory to get data, ***rdtsc*** is executed speculatively, then the result of rdtsc execution, which should have been placed in ***EDX:EAX*** is limited to three bits from ***EAX***, and memory from the pool, that was created earlier is accessed with the page offset, based on that value plus 32. This means that there are only one of 8 pages (there 256 pages in the buffer) will be cached if the speculative instructions are executed. The number of page with the lowest access time is added to statistics. Then all cache in the memory region is flushed.

For a successful attack this is done many times (10000 cycles were used in our case). In the end the number of times pages that were not designated for the check was computed and based on that value the decision is made. In virtualized environments where VMEXIT on ***rdtsc*** was enabled, the percentage of time non-designated areas are hit spans from 50% to 99%. On non-virtualized systems it is less than one percent. Mac OS, Ubuntu, Debian and Windows non-virtualized hosts were used for testing. Ubuntu, Debian and Windows were used as guest environments.

4. **Explanation of the attack**

CPU decides that ***rdtsc*** would take too long to execute in a VM and doesn't execute it unless the flow reaches it directly. In a virtualized environment ***rdtsc*** and instructions directly following it are not executed speculatively, and in non-virtualized they are.

5. **Conclusions and future work**

There already existed several methods for checking Virtualization, however they mostly relied heavily on using rdtsc for timing, leaving them vulnerable to a smart hypervisor, that could possibly fake the values. While this attack in its simplest form is vulnerable to that too, it can be changed to operate on limited memory regions during one cycle. If the memory regions are deterministically changed between cycles in a certain manner, it becomes infeasible for the hypervisor to try faking the memory accesses as if they were cached, because it doesn't know which ones should have been cached. A POC will be created that uses this property.

This creates and interesting observation: if the ***rdtsc*** creates a VMEXIT, the attack introduced in this article can show the presence of virtualization, while many previous attacks can be mitigated, and if there is no VMEXIT on ***rdtsc***, previous attacks are useful.